Recorp: Receiver-oriented Policies for Industrial Wireless Networks

RYAN BRUMMET, MD KOWSAR HOSSAIN, OCTAV CHIPARA, TED HERMAN, and DAVID E. STEWART, University of Iowa

Future Industrial Internet-of-Things (IIoT) systems will require wireless solutions to connect sensors, actuators, and controllers as part of high data rate feedback-control loops over real-time flows. A key challenge in such networks is to provide predictable performance and adaptability in response to link quality variations. We address this challenge by developing RECeiver ORiented Policies (Recorp), which leverages the stability of IIoT workloads by combining offline policy synthesis and run-time adaptation. Compared to schedules that service a single flow in a slot, Recorp policies share slots among multiple flows by assigning a coordinator and a list of flows that may be serviced in the same slot. At run-time, the coordinator will execute one of the flows depending on which flows the coordinator has already received. A salient feature of Recorp is that it provides predictable performance: a policy meets the end-to-end reliability and deadline of flows when the link quality exceeds a user-specified threshold. Experiments show that across IIoT workloads, policies provided a median increase of 50% to 142% in real-time capacity and a median decrease of 27% to 70% in worst-case latency when schedules and policies are configured to meet an end-to-end reliability of 99%.

CCS Concepts: • **Networks** → **Network protocols**; **Network algorithms**; **Network dynamics**; Network reliability; **Cyber-physical networks;**

Additional Key Words and Phrases: Wireless communication, TDMA, reliability

ACM Reference format:
Ryan Brummet, Md Kowsar Hossain, Octav Chipara, Ted Herman, and David E. Stewart. 2021. Recorp: Receiver-oriented Policies for Industrial Wireless Networks. *ACM Trans. Sen. Netw.* 17, 4, Article 44 (July 2021), 32 pages.
https://doi.org/10.1145/3460618

1 INTRODUCTION

Industrial Internet-of-Things (IIoT) systems are gaining rapid adoption in process control industries such as oil refineries, chemical plants, and factories. In contrast to prior work that has focused primarily on low-data rate or energy-efficient applications, we are interested in the next generation of smart factories expected to use sophisticated powered sensors such as cameras, microphones, and accelerometers (e.g., References [12, 23, 28]). Since such applications will require higher data rates, we need to develop a versatile wireless solution to connect them with actuators

This work is funded in part by NSF under Grant No. CNS-1750155.
Authors' address: R. Brummet, Md K. Hossain, O. Chipara, T. Herman, and D. E. Stewart, The University of Iowa, Iowa City, IA 52242-1419; emails: {ryan-brummet, mdkowsar-hossain, octav-chipara, ted-herman, david-estewart}@uiowa.edu.

1550-4859/2021/07-ART44 $15.00
https://doi.org/10.1145/3460618

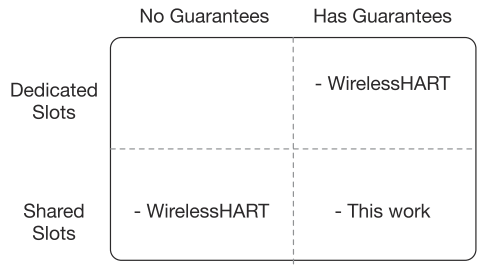


Fig. 1. Design space of wireless control solutions.

and controllers as part of feedback-control loops over multihop *real-time flows*. A practical solution must meet the following two requirements: (1) must support high data rates, and (2) support real-time communication over multiple hops. Both requirements must be met, notwithstanding significant variations in the quality of wireless links common in harsh industrial environments [8, 17].

Let's examine whether WirelessHART can meet the requirements of the next generation of IIoT applications. WirelessHART is the state-of-the-art standard for industrial wireless communication and has successfully provided high reliability in a broad range of industrial settings. At the heart of WirelessHART is **Time Slotted Channel Hopping (TSCH)**—a MAC layer that combines **Time Division Multiple Access (TDMA)** and channel-hopping in a mesh network. The TSCH data plane relies on a centralized network manager to generate routes and a transmission schedule for all the flows in the network. The schedule is represented as a two-dimensional scheduling matrix that specifies the time and frequency of each transmission. TSCH supports both real-time and best-effort traffic by using two scheduling strategies whose trade-offs are shown in Figure 1.

To support real-time traffic, a transmission is assigned to a *dedicated entry* in the scheduling matrix and, at run-time, the transmission is performed without contention. The reliability of real-time traffic is ensured by using retransmissions and channel hopping. The number of retransmissions allocated is usually determined based on the worst-case quality of a link as required to tolerate significant variations in link quality. Since the scheduling matrix cannot be updated at the rate at which the link quality varies, the only run-time adaption available is to cancel a link's retransmissions when an acknowledgment is received. As a result, it is common for a significant number of slots to remain unused when a packet is relayed successfully to the next hop before exhausting a link's allocated retransmissions. As demonstrated by the experiments in Section 6, protocols that use dedicated entries cannot handle higher data rates efficiently.

In contrast, best-effort traffic is supported by having multiple transmissions assigned to a *shared entry* in the scheduling matrix. At run-time, contention-based techniques are used to arbitrate which transmissions will be performed. Shared entries provide more opportunities for locally adapting what transmissions may be performed, resulting in more efficient use of network resources. Unfortunately, there are no current techniques to effectively analyze the network's performance when shared entries are used. The open research question, and the focus of this article, is whether *it is possible to use shared entries to support higher throughput and respond more effectively to changes induced by link quality variations while providing performance guarantees.*

To answer this question, we propose ***RECeiver ORiented Policies (Recorp)***—a new data plane that provides higher performance and agility than traditional scheduling solutions that do not use shared entries. We exploit the typical characteristics of the industrial setting to obtain improvements in network capacity and latency while providing predictability under prescribed link variability. Specifically, our approach has the following features:

- Since IIoT workloads consist of sets of real-time flows that are stable for long periods of time, we compute offline Recorp policies and disseminate them to all nodes. Recorp policies assign a coordinator and a list of candidate flows for each entry in the scheduling matrix. Only one of the candidate flows will be executed at run-time, depending on which flows the coordinator has already received. The benefit of this approach is that it enables flows to be dynamically executed in an entry depending on the successes and failures of transmissions observed at run-time. As a consequence, Recorp policies can handle variations in link quality more effectively than schedules.
- We propose a novel link model in which the quality of the links can vary *arbitrarily* within an interval from slot to slot. Our model is motivated by current guidelines for deploying wireless IIoT networks (e.g., Reference [29]), focusing on ensuring that communication links have a minimum link quality. The proposed model is well-suited to industrial settings where link quality may vary widely over short time scales.
- In contrast to best-effort entry sharing approaches that provide no performance guarantees, we ensure that a constructed Recorp policy will meet a user-specified reliability and deadline constraint for each flow as long as the quality of all (used) links exceeds a minimum link quality as specified by our model.

We demonstrate the effectiveness of Recorp through testbed measurements and simulations. When schedules and Recorp policies are configured to meet the same target end-to-end reliability of 99%, empirical results show that Recorp policies increased the median real-time capacity by 96% for a data collection workload. Furthermore, the performance bounds derived analytically were safe: Recorp policies met all end-to-end reliability and deadline constraints when the minimum link quality exceeded a user-specified level of 70%. Larger-scale multihop simulations that consider two topologies indicate that across typical IIoT workloads, policies provided a median increase of 50% to 142% in the real-time capacity as well as a median decrease of 27% to 70% in worst-case latency.

The remainder of the article is organized as follows. Section 2 describes the problem formulation and informally introduces Recorp policies and the challenges associated with their synthesis. Section 3 describes Recorp's system and network models, while Section 4 introduces Recorp's reliability model. Recorp is described in Section 5. Simulation and testbed experiments evaluating Recorp's performance against representative protocols using both dedicated and shared entries are included in Section 6. Section 7 describes how Recorp handles network dynamics, aperiodic traffic, and energy efficiency. Recorp is placed in the context of related work in Section 8. We conclude the article in Section 9.

2 PROBLEM FORMULATION

In this section, we start by considering the problem of building real-time protocols from a fresh perspective, discuss how this perspective opens new opportunities for optimization, and then informally introduce Recorp policies while highlighting the challenges of their synthesis.

Optimization Problem: We consider supporting real-time and reliable communication as a sequential decision problem. In each slot, the offline policy synthesis procedure uses the current estimate of the network state to select the actions performed in the current slot. Then, the estimated network state is updated to reflect the impact of those actions. In this article, we limit our attention to myopic (or greedy) policies that maximize the number of flows that may be executed in a slot while providing prioritization based on the flows' statically assigned priorities. A myopic policy selects the optimal actions over a time horizon of one slot, but those decisions may be suboptimal over longer horizons. Our choice is motivated by the simplicity of myopic policies that can be

synthesized efficiently. The unique aspects of Recorp policies are what actions may be performed in a slot and how the network state is represented.

Intuition: Schedules and policies differ in the information they use as part of the offline scheduling and synthesis process. Consider a star topology with three nodes where the base station is the receiver of two incoming flows F_0 and F_1. Both flows are released at the beginning of slot 0 with flow F_0 having a higher priority than flow F_1. Since F_0 and F_1 share the same receiver, only one of them can transmit in the first slot without conflict. In slot 0, both schedules and policies assign and execute (at run-time) F_0 to enforce prioritization.

Schedules and policies differ in how they account for the outcome of F_0's transmission. At run-time, the network is in one of two states, depending on the outcome of F_0's transmission: either F_0's data was relayed successfully to the base station, or it was not. Scheduling approaches ignore this information and assign a fixed number of retransmissions for F_0, regardless of whether these retransmissions are successful or not at run-time. However, when we capture both possible outcomes, there are new opportunities for optimization. Ideally, we would like to transmit F_1 if F_0 has succeeded or otherwise retransmit F_0. Surprisingly, we can achieve this behavior (which is impossible for scheduling approaches): Offline, both flows F_0 and F_1 are assigned to be *candidates* to be executed in slot 1. At run-time, the base station will track the flows from which it has received packets in the previous slots. As a result, it will know whether F_0 was successful or not at the end of slot 0, i.e., the star network's precise state. Using this information, in slot 1, the base station can request F_1's packet if it has already received F_0 or, otherwise, it can request a retransmission for F_0. We say that F_0 and F_1 *share* slot 1 as either flow may execute at run-time depending on the observed successes and failures.

Actions: This approach can be generalized to multi-hop scenarios by observing that any node with multiple flows routed through it can act as a coordinator for those flows, not just a base station in a star topology. A Recorp policy is represented as a matrix whose rows indicate channels and columns indicate slots. In each entry of the matrix, a Recorp policy may include, at most, one pull. A *pull* has two arguments: a *coordinator* and a *service list*. A pull is executed by a coordinator that can dynamically request data (i.e., a `pull`, henceforth) from a service list of flows depending on the outcome of previous transmissions. The synthesis procedure determines the nodes that will be coordinators and the composition of the service list, both of which can change from slot to slot. At run-time, a coordinator executing a pull requests the packet of the first flow in the service list from which it has not yet received the packet. The adaptation mechanism is localized, lightweight, and does not require carrier sense.

State Estimation: A challenge to synthesizing policies is to estimate the network's state as pulls are performed. Specifically, we need to know the likelihood that a flow's packet is located at a specific node in a given slot. Knowing this information *offline* is challenging, because the quality of a link is probabilistic, and the likelihood of a successful transmission varies from slot to slot. To address this challenge, we propose a **Threshold Link Reliability (TLR)** model. We model the quality of a link $LQ_i(t)$ in slot t used by flow i as a Bernoulli variable. TLR allows the quality of the link to change arbitrarily from slot-to-slot as long as it exceeds a minimum value m (i.e., $LQ_i(t) \geq m \; \forall \; t$). We will show it is possible to provide guarantees on the performance of Recorp when all links follow the TLR model.

Scalability: Another significant challenge in synthesizing policies is avoiding the state explosion problem. The critical decision is how to balance the trade-off between the expressiveness of policies, the performance improvements they provide, and the scalability of the synthesis procedure. Cognizant of these trade-offs, we make two important design choices: (1) We limit nodes to operating on their local states such that their decisions are independent of the state of other nodes. As a consequence, the probabilities of packets being forwarded across links of a multi-hop flow are

independent. This property reduces the number of states maintained during synthesis, since it is sufficient to capture the interactions of flows locally at each node rather than globally across the network. (2) The synthesis procedure incrementally constructs policies in a slot-by-slot manner using a builder and an evaluator. The builder casts the problem of determining the pulls performed in the current slot as an **Integer Linear Program (ILP)**. In turn, the evaluator applies each pull selected by the builder to the system state and tracks the state as it evolves from slot to slot. The iterative nature of the synthesis procedure improves its scalability as it suffices to maintain only the states associated with the current slot.

3 SYSTEM MODEL

We base our network model on WirelessHART as it is an open standard developed specifically for IIoT systems with stringent real-time and reliability requirements [2]. A network consists of a *base station* and tens of *field devices*. Recorp is best-suited for applications that require high data rates and have a backbone of grid-powered nodes to carry this traffic (e.g., References [4, 7, 10]).

A centralized *network manager* is responsible for synthesizing policies, evaluating their performance, and distributing them across the network. The field devices form a time-synchronized wireless mesh network that we model as a graph $G(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ represent the devices (including the base station) and wireless links. The network can be synchronized using a high-accuracy time synchronization protocol designed for wireless sensor networks (see Reference [32] for a survey). We will initially assume that the communication graph remains fixed while each link has a minimum link quality. In Section 7, we will discuss how Recorp can handle node failures and topology changes, such as adding and removing nodes, by distributing new policies. The network maintains two trees, an *upstream tree* and *downstream tree*, for packet routing to and from the base station, respectively. We assume that both upstream and downstream trees are spanning trees consistent with source routing in WirelessHART.

At the physical layer, WirelessHART adopts the 802.15.4 standard with up to 16 channels. This article focuses on receiver-initiated communication, where a node requests data from a neighbor and receives a response within the same 10 ms slot.

We use *real-time flows* as a communication primitive. The following parameters characterize a real-time flow F_i: phase σ_i, period P_i, deadline D_i, end-to-end target reliability requirement T_i, and static priority i where lower values have higher priority. The kth instance of flow F_i, $J_{i,k}$, is released at time $r_{i,k} = \phi_i + k * P_i$ and has an absolute deadline $d_{i,k} = r_{i,k} + D_i$. We assume $D_i \leq P_i$, which implies only one instance of a flow is released at a time. Consequently, to simplify the notation, we will use J_i to refer to the instance of flow F_i that is currently released. The variable $\mathcal{F}$ denotes the set of flows in the network. A flow i has a forwarding path Γ_i that is used by all of its instances. During the execution of an instance, only one of the links on the Γ_i is active and considered for scheduling. We will use the notation $LQ_i(t)$ to refer to the link quality of the currently active link at time t.

A Recorp policy π is a scheduling matrix whose number of slots is equal to the hyperperiod of the flow's periods. The policy may be represented as a two-dimensional matrix such that the rows indicate channels, the columns indicate slots, and the entries that represent actions. An action may be either a pull or a sleep. A policy is well-formed if it satisfies the following constraints: (1) Each node transmits or receives at most once in an entry to avoid intra-network interference. (2) The hop-by-hop packet forwarding precedence constraints are maintained such that senders receive packets before forwarding them. (3) Nodes do not perform consecutive transmissions using the same channel. (4) Each flow instance is delivered to its destination before its absolute deadline and meets its reliability constraint.

Table 1. Summary of Key Notations

Description	Symbol
Set of nodes	$\mathcal{N}$
Set of flows	$\mathcal{F}$
Flow i	F_i
Period of flow i	P_i
Deadline of flow i	D_i
Phase of flow i	ϕ_i
Target end-to-end reliability of flow i	T_i
Path of flow i	Γ_i
Quality of the active link of flow i	LQ_i
Instance k of flow i	$J_{i,k}$ (or simply J_i)
Release time of $J_{i,k}$	$r_{i,k}$
Absolute deadline of $J_{i,k}$	$d_{i,k}$
Link quality of the active link of $J_{i,k}$	LQ_i
Policy	π
Service list of the pull in slot t (and channel c)	$srv(t)$ (or $srv(t,c)$)
Set of all possible states	Ψ
Transition matrix	$\mathcal{M}$
Reliability of instance J_i	R_i
Lower-bound on reliability of J_i	$\widehat{R}_i$

4 RELIABILITY MODEL

The wireless communications community has developed a wide range of probabilistic models to model link quality (e.g., References [19, 25]). Examples span the complexity-accuracy trade-off from simple models such as Gilbert-Elliott [19] to more complex models that use multi-level Markov Chains (e.g., Reference [25]) to distinguish between the short-term and long-term behavior of wireless links. However, these models usually focus on the "average-case" behavior of links. Guarantees on the end-to-end reliability of flows should hold even as links deviate from their average-case behavior. Furthermore, a practical model must require little tuning, preferably having reasonable default values for its parameters that fit the rules-of-thumb engineers use to deploy real wireless networks.

To address the above challenges, we propose the TLR model. We model the likelihood that a single pull for flow i is successful (including both the pull request and the response containing the data) as a Bernoulli variable $LQ_i(t)$. We assume that consecutive pulls performed over the same or different links are independent. Empirical studies suggest that this property holds when channel hopping is used [21, 24]. A minimum **Packet Delivery Rate (PDR)** m lower bounds the values of $LQ_i(t)$ such that $m \leq LQ_i(t)\ \forall i \in \mathcal{F}, t \in \mathbb{N}$. A strength of TLR is that aside from the lower bound m on link quality, we make *no assumptions regarding how the quality of a link varies from slot to slot.* This characteristic makes TLR widely applicable to networks experiencing significant link quality variations. TLR can be integrated with existing guidelines for deploying IIoT wireless networks. For example, Emerson engineers suggest that WirelessHART networks should be deployed to provide a minimum link quality between 60–70% [29]. Accordingly, in this article, we set m to either 60% or 70%.

On a more technical note, it is important to note that TLR does not require the transmissions in an actual network deployment to be independent—we only require that there is a TLR model that lower bounds the behavior of the deployed network. Specifically, we require that a Bernoulli

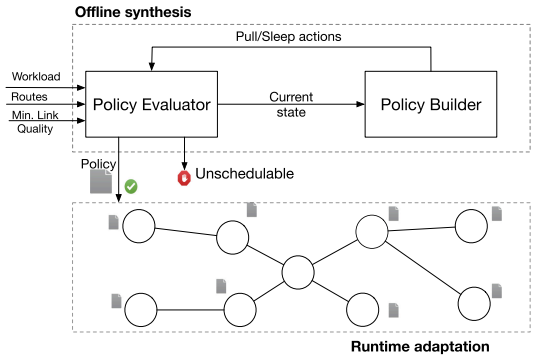


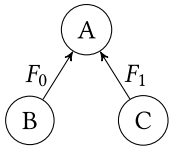
Fig. 2. Design of Recorp.

distribution *lower bounds* the distribution of consecutive packet losses in the network. Thus, by selecting an appropriate value for m, it is possible to find a model for which the assumption of independence holds, albeit at the cost of increased pessimism regarding the quality of links.

The end-to-end reliability R_i of a flow i depends on both the likelihood of successfully relaying a packet over the links of its path as well as the links of other flows it shares entries with. For instance, returning to our running example, the probability the packet released by F_1 reaches its destination is dependent not only on the quality of its link but also F_0's link, since F_1 is conditionally attempted depending on the success of F_0. One might assume that finding a lower bound on R_i under the TLR model only requires considering the case when all links exhibit their worst link quality in all slots (i.e., $LQ_i(t) = m \; \forall i \in \mathcal{F}, \; t \in \mathbb{N}$). While we will show that this approach provides a safe lower bound for Recorp policies, this property does not hold for *all* policies that use shared slots. Consider, for example, the two flows F_0 and F_1. Suppose these flows are scheduled using the following simple (non-Recorp) policy. In the first slot, F_0 will be executed. In the next slot, F_1 will be executed only if F_0 failed in the first slot; otherwise, the base station sleeps. Under this policy, the probability that F_1 is attempted will decrease as the link quality increases, since increasing the link's quality will increase the probability that F_0 is successful in the first slot. As a consequence, the end-to-end reliability of F_1 will drop as the link becomes more reliable. Therefore, for policies such as Recorp that share slots, it is essential to prove that they do not exhibit such pathological behavior. Theorem 2 demonstrates that Recorp policies do not exhibit this behavior.

5 DESIGN

Recorp is a practical and effective solution for IIoT applications that require predictable, real-time, and reliable communication in dynamic wireless environments (see Figure 2). Central to our approach is Recorp policies. The policy synthesis procedure runs on the network manager and has as inputs the workload, routing information, and a user-specified minimum link quality threshold m. If the synthesis procedure is successful, then the constructed policy guarantees probabilistically that all flows will meet their real-time and reliability constraints as long as the quality of all links meets or exceeds m. The synthesis procedure fails when the workload is unschedulable, i.e., when a policy that meets both the real-time and reliability constraints of all flows cannot be found. Note that this case is unlikely to arise in practice, since an application's workload specification is known *a priori*, and the designer can validate that the workload remains unschedulable during the system's deployment. If the synthesis procedure is successful, then the manager disseminates the generated policy to all nodes. During the operation of the network, some links may fall below the minimum link quality threshold m. Since Recorp provides no guarantees under this



(a) Topology

	0	1	2	3
Spec	$TX_B(J_0)$	$TX_B(J_0)$	$TX_C(J_1)$	$TX_C(J_1)$
Trace 1	(BA)		(CA)	
Trace 2	(BA)	x (BA) x	(CA)	
Trace 3	(BA)		(CA)	x (CA) x

(b) Schedule

	0	1	2	3
Spec	$PL_A(J_0)$	$PL_A(J_0,J_1)$	$PL_A(J_0,J_1)$	$PL_A(J_1)$
Trace 1	(BA)	(CA)		
Trace 2	(BA)	(BA)	(BA)	(CA)
Trace 3	(BA)	(CA)		

(c) Recorp policy

Fig. 3. A schedule and policy for the topology shown in Figure 3(a) are constructed. At run-time, schedules and policies behave differently depending on observed successes (green background) or failures (red background). The traces show how schedules and policies adapt run-time behavior in response to successes and failures. Notably, the schedule drops packets in traces 2 and 3 (indicated by white "x"-es) while the policy drops no packets.

regime, a new policy should be constructed after either changing the flows' routes to avoid low-quality links or by lowering m.

The separation between offline synthesis and run-time adaptation is essential to building agile networks. The run-time adaptation is lightweight: When a node is the coordinator of a pull, it can execute any of the flows included in its service list without requiring global consensus. In contrast, policy synthesis is computationally expensive and ensures the global invariant that no transmission conflicts occur regardless of coordinators' local decisions.

We will formalize the semantics of Recorp policies and discuss their run-time adaptation mechanism in Section 5.1. After, we will consider synthesizing Recorp policies in a scalable manner. We will start by considering the problem of synthesizing policies for a data collection workload in a star topology in Section 5.2. In Section 5.3, we will extend our approach to handle general workloads and topologies.

5.1 Recorp Policies and Their Run-time Adaptation

A Recorp policy is represented as a scheduling matrix with a sleep or a pull action in each entry. A sleep action indicates that no action is taken in a slot and channel. A pull has two arguments: a *coordinator* and a *service list*. The coordinator is the node that executes the action at run-time, and the service list includes the instances that *may* be executed in that slot and channel. The instances in the service list are ordered according to the priority of their flows. At run-time, only one of the candidate flows in the service list will be executed. Any node can become a coordinator, and the coordinators can change from slot to slot. The execution of a policy is cyclic, with nodes returning to the policy's beginning upon reaching its end.

A coordinator executes a pull at run-time by considering the instances in the service list in priority order. For each instance, the coordinator checks whether it has received the instance's packet. If the coordinator has already received the instance, then it will consider the next instance in the service list. Otherwise, it will request the instance's packet from the coordinator's neighbor through which the instance is routed. Upon receiving a request, the neighbor may or may not have the packet (the latter case can happen when the packet was dropped at a previous hop). If the neighbor has the packet, then it includes it in its response to the coordinator. Otherwise, the neighbor marks the packet as dropped in its response. In response to receiving either response, the coordinator marks the instance as successfully executed. The invariant maintained by the execution of a pull is, *at most, one instance from the service list is executed in a slot.* Note that the request or the response may not be delivered, since links are unreliable. We account for this by having an instance be included in the service list of several pulls performed by the coordinator. As discussed

in Section 5.2, an instance is included in the service list of sufficient pulls to meet the flow's target end-to-end reliability, given the TLR's minimum reliability threshold.

The proposed adaptation mechanism is sufficiently lightweight to run within 10 ms slots, as specified by WirelessHART. The memory usage is proportional to the number of flows routed through a node, which is small. Equally important, the adaptation mechanism does not employ carrier sensing and, instead, relies on receiver-initiated pulls.

To illustrate the differences between Recorp policies and schedules, consider a star topology (see Figure 3(a)). In this example, two flows—F_0 and F_1—relay data from B and C to the sink A. In slot 0, instances J_0 and J_1 are released from flows F_0 and F_1, respectively. WirelessHART requires the construction of a schedule with two transmissions for each instance (see Figure 3(b)). Three traces that differ in the pattern of packet losses observed at run-time are also included in the figure. The only run-time adaptation mechanism available in schedules is to cancel scheduled transmissions whose data has already been delivered. The notation $\text{TX}_B(J_0)$ indicates that B transmits J_0's packet to A. The synthesized Recorp policy is shown in Figure 3(c) and uses the notation $\text{PL}_A(J_0, J_1)$ to indicate a pull with A as the coordinator and $\{J_0, J_1\}$ as the service list.

To highlight several differences between policies and schedules, consider trace 2, where there are failures in slots 0 and 1. For this trace, the schedule included in Figure 3(b) cannot successfully deliver J_0's packet, because it is allocated only a single retransmission. In contrast, the Recorp policy included in Figure 3(c) can successfully deliver J_0's packet. The policy includes J_0 in the service list of the pulls in slots 0, 1, and 2. At run-time, J_0's transmission in slots 0 and 1 fails, but J_0 will be delivered in slot 2. In slot 3, the policy successfully executes J_1. A similar scenario is included in trace 3, where J_1's packets cannot be delivered by schedules but are successfully delivered using a Recorp policy. Traces 2 and 3 highlight the flexibility of Recorp policies to improve reliability by dynamically reallocating retransmissions based on the successes and failures observed at run-time.

A key property of the run-time adaptation mechanism that we will leverage during policy synthesis is the following:

Theorem 1. *The execution of Recorp actions on a node is not affected by the actions of other nodes.*

Proof. Consider the execution of a pull by a node R. A pull's behavior depends on what instances are included in the service list and the local state of the node. Since the service list is fixed once the policy is constructed, the only way another node may affect R's state is by directly modifying its state, which does not happen. □

5.2 Synthesizing Recorp Policies for Data Collection on Star Topologies

As a starting point, let us consider the problem of constructing Recorp policies for a star topology where all flows have the base station as the destination (see Figure 3(a) for an example). This setup simplifies the synthesis of policies in two regards: (1) The base station will be the coordinator of all pulls. Therefore, we only have to focus on determining the service list of each pull. (2) Since all flows have the base station as the destination, there will be no transmission conflicts, and a (different) single channel can be used in each slot. We will generalize our approach to general multi-hop topologies and workloads in the next section.

The policy synthesis procedure involves two key components—an evaluator and a builder (see Figure 2). The policy is synthesized incrementally by alternating the execution of the builder and evaluator in each slot.

- The builder determines the pulls that will be executed in each slot. The builder maintains an active list that contains all of the instances that have been released but have *not* yet met their end-to-end reliability. In a slot t, the builder checks whether an instance $J_{i,k}$ is released

(i.e., when $r_{i,k} = t$) and, if this is the case, $J_{i,k}$ is added to the active list. If the active list is not empty in t, then a pull having the base station as coordinator and the instances in the active list as its service list is assigned in the entry t of the matrix.
- The evaluator maintains the likelihood that each instance in the active list has been delivered to the base station. The probabilities are updated incrementally to reflect the execution of the pull provided by the builder in slot t.
- At the end of slot t, the builder removes all instances whose reliability exceeds their end-to-end reliability targets from the active list.

In the remainder of the section, we will answer the question of how to estimate the reliability of flows given the sequence of pulls determined by the builder. This problem can be modeled at a high level as a **Markov Decision Process (MDP)** whose transitions depend on the likelihood of successfully executing pulls. Let Ψ be the set of all possible states. A state s ($s \in \Psi$) is represented as a vector of size $|\mathcal{F}|$, where the i^{th} entry represents the state of instance J_i. The state of an instance J_i may be S or F, indicating whether the base station requested J_i's data and received a reply successfully. The reply may either include a flow's packet or an indication that it has been dropped on a previous hop. A direct encoding of this information would require $O(2^{|\mathcal{F}|})$ states, which is not practical when there are numerous flows. To avoid state explosion, we propose the following mechanism. We bound the length of the active list maintained by a coordinator. This requires a simple modification to the builder: an instance is added to the active list until it reaches the user-specified maximum size. The additional instances that are released when the active list is full are added to an `inactive` list. The `inactive` list includes instances that are released but not yet active. When an instance completes, the size of the active list decreases by one, and the highest priority instance from the `inactive` list is moved to the active list.

With this modification, the maximum number of states a coordinator maintains is reduced to $O(2^{|\text{active list}|})$. Additionally, we observe that the likelihood an instance is executed depends on its index in the service list. If the index of an instance in the service list exceeds 3 or 4, then the instance is unlikely to be executed. Accordingly, we also cap the maximum size of the service list. The service list of a pull is then a subset of the active list. In our experiments, we constrain $|\text{active list}| \leq 10$ and $|\text{service list}| \leq 4$ except where otherwise stated.

End-to-end Reliability Using Instantaneous Link Quality: Let us start by deriving a method for computing the end-to-end reliability of flows under the assumption that there is an omniscient oracle that can provide the instantaneous probability of a successful pull for all links in a slot t. We will use the notation $\boldsymbol{LQ}_t$ to represent the link quality of all links in slot t. Later, we will relax this requirement by constraining links to follow the TLR model i.e., their link quality is lower bounded by m (i.e., $LQ_i(t) \geq m$). Under this assumption, we will show that the worst-case end-to-end reliability of a flow occurs when the quality of all links is equal to m in all slots.

The actions of the MDP are the pulls that the builder assigns in each slot. Initially, the system is in a state s_0, in which the base station has not received the data from any of the flows. Consider the execution of a pull with service list srv in slot t. To account for the impact of executing the pull on the state of the system, we construct a transition matrix $\mathcal{M}_{srv(t)}$ of size $2^{|\text{active list}|} \times 2^{|\text{active list}|}$ using Algorithm 1. Let J_i be an instance included in the service list srv (not necessarily as the head of the list). According to the semantics of pulls, J_i will be executed in any *current* state where the i^{th} entry of the vector is a failure (i.e., $current[i] = \text{F}$) and the execution of all instances J_j with higher priority than J_i in the service list srv have already succeeded (i.e., $current[j] = \text{S}$). From such a *current* state, there are two possible outgoing transitions depending on whether the pull is successful or not. If the execution of J_i fails, then the system remains in the same state (see line 7, Algorithm 1). Accordingly, the entry $\mathcal{M}_{srv(t)}[current, current]$ is set to $1 - LQ_i(t)$,

ALGORITHM 1: Computes the transition matrix $\mathcal{M}_{srv(t)}$ given the service list srv of a pull and a snapshot of current link $\boldsymbol{LQ}_t$

```
1:  Procedure BuildTransitionMatrix(srv(t), LQ_t)
2:      M_srv(t) = I
3:      for current in Ψ do
4:          for J_i in srv do
5:              Let i be the flow id of J_i
6:              if current[i] = F then
                    /* the execution fails                                   */
7:                  M_srv [current, current] = 1 - LQ_i(t)
                    /* the execution is successful                           */
8:                  next = onSuccess(current, i)
9:                  M_srv(t) [current, next] = LQ_i(t)
10:                 break
11:     return M_srv(t)
12: Procedure onSuccess(state, i)
        /* the next_state is the same as current except for the entry for J_i becomes S */
13:     next_state[j] = state[j]  ∀j ≠ i
14:     next_state[i] = S
15:     return next_state
```

where $LQ_i(t)$ is the probability of performing a successful pull over the link used by flow i in slot t. Conversely, if the execution of J_i succeeds, the system transitions from the *current* state to a *next* state. The entries of the *current* and the *next* states are the same, except for the entry associated with the J_i element for which $next[i] = \mathsf{S}$ (see line 12, Algorithm 1). In this case, we set $\mathcal{M}_{srv(t)}[current, next] = LQ_i(t)$. If a sleep is assigned slot t, then the state of the system does not change.

After executing t pulls, the probability of each state is given by the vector $\boldsymbol{P}_t$:

$$\boldsymbol{P}_t = \boldsymbol{s}_0^T \mathcal{M}_{srv(0)} \mathcal{M}_{srv(1)} \cdots \mathcal{M}_{srv(t)}, \tag{1}$$

where $\boldsymbol{s}_0$ is the initial state of the system and $\mathcal{M}_{srv(t')}$ is the transition matrix associated with the pull that has $srv(t')$ as its service list and is executed in slot t' ($0 \leq t' \leq t$). Equation (1) describes the evolution of the system as a discrete-time **Markov Chain (MC)** that is parametric and time inhomogeneous. The structure of $\mathcal{M}_{srv(t')}$ depends on the service list and its values depends on the quality of all links in slot t'.

The end-to-end reliability R_i of instance J_i after executing t pulls is computed by summing up the probability of each state s ($s \in \Psi$) such that $s[i]$ is S. Leveraging the properties of matrix multiplication, R_i may be written as

$$R_{i,t} = \boldsymbol{P}_t \boldsymbol{\chi}_i, \tag{2}$$

where $\boldsymbol{\chi}_i$ is a vector such that $\boldsymbol{\chi}_i[k] = 1$ for any state s such that $s[k] = \mathsf{S}$ and $\boldsymbol{\chi}_i[k] = 0$ otherwise.

End-to-end reliability under TLR: Computing $R_{i,t}$ requires that we know the instantaneous quality of all links in any slot t. It is infeasible to have access to this information during the synthesis of a policy. In the following, we will derive a lower bound $\widehat{R}_{i,t}$ on $R_{i,t}$. To this end, we will construct a new MC with transition matrix $\widehat{\mathcal{M}}_{srv(t)}$ that is computed by considering each transition matrix $\mathcal{M}_{srv(t)}$ and replacing each link quality variable $LQ_i(t)$ with its lower-bound m. We

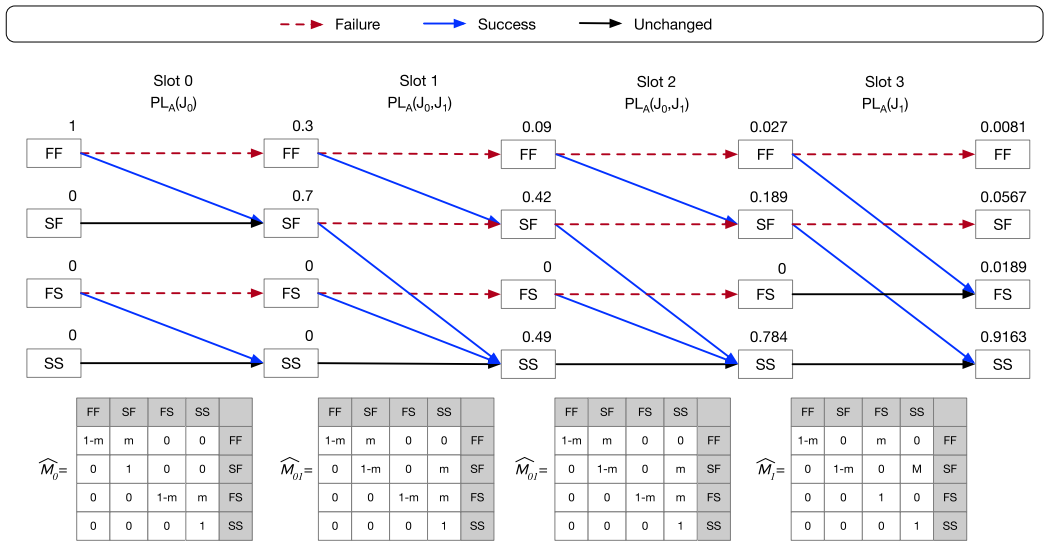


Fig. 4. Estimating the state of the network and lower-bounds on the end-to-end reliability.

claim that a lower bound on the end-to-end reliability of a flow $R_{i,t}$ is

$$R_{i,t} \geq \widehat{R}_{i,t} = \widehat{\boldsymbol{P}}_t \boldsymbol{\chi}_i = \boldsymbol{s}_0^T \widehat{\mathcal{M}}_{srv(0)} \widehat{\mathcal{M}}_{srv(1)} \cdots \widehat{\mathcal{M}}_{srv(t)} \boldsymbol{\chi}_i. \tag{3}$$

The following theorem implies that to compute a lower-bound on the reliability of a flow, it is sufficient to consider only the case when all links perform their worst.

Theorem 2. *Consider a star topology that has node A as a base station and a set of flows $\mathcal{F} = \{F_0, F_1, \ldots F_N\}$ that have A as destination. Let $LQ_0(t)$, $LQ_1(t)$, $\ldots LQ_N(t)$ be the quality of the links used by each flow in slot t such that $m \leq LQ_i(t) \leq 1$ for all flows F_i ($F_i \in \mathcal{F}$) and all slots t ($t \in \mathbb{N}$). Under these assumptions, the reliability $R_{i,t}$ of an instance J_i after executing t pulls of the Recorp policy π is lower bounded by $\widehat{R}_{i,t}$.*

Proof. See Section A. □

Let us return to our running example of the construction and execution of the policy shown in Figure 3(c). In Figure 4, we will illustrate how the end-to-end reliability of flows will be estimated for this example. The workload includes two flows—F_0 and F_1—with phases $\phi_0 = 0$ and $\phi_1 = 1$. Accordingly, instances J_0 and J_1 are released in slots 0 and 1. We will evaluate the estimated state of the network $\widehat{\boldsymbol{P}}_t$ and the lower-bounds on the reliability of each flow as the policy is executed. Given that the workload involves only two flows, the possible states of the systems are $\Psi = \{\mathsf{FF}, \mathsf{SF}, \mathsf{FS}, \mathsf{SS}\}$. Each state encodes whether the base station A has received the data of J_0 and J_1. In any slot t, the probability vector $\widehat{\boldsymbol{P}}_t$ is the likelihood that the network is in a state FF, SF, FS, and SS (in that order). The lower bound on the reliability of instance J_0 is $\widehat{R}_{0,t} = \widehat{\boldsymbol{P}}_t[\mathsf{SF}] + \widehat{\boldsymbol{P}}_t[\mathsf{SS}] = \widehat{\boldsymbol{P}}_t \chi_0$, where $\boldsymbol{\chi}_0 = [0, 1, 0, 1]$. Similarly, $\widehat{R}_{1,t} = \widehat{\boldsymbol{P}}_t[\mathsf{FS}] + \widehat{\boldsymbol{P}}_t[\mathsf{SS}] = \widehat{\boldsymbol{P}}_t \boldsymbol{\chi}_1$, where $\boldsymbol{\chi}_1 = [0, 0, 1, 1]$.

Initially, the system is in state $\boldsymbol{s}_0 = [1, 0, 0, 0]^T$ i.e., $\boldsymbol{s}_0[\mathsf{FF}] = 1$ and the likelihood of the remaining states is zero. The action $\mathrm{PL}_A(J_0)$ is executed in slot 0. The evaluator constructs the matrix $\widehat{\mathcal{M}}_0$ to account for the impact of executing the pull on the state of the system. After executing the pull, the state of the network is $\widehat{\boldsymbol{P}}_0 = \boldsymbol{s}_0^T \widehat{\mathcal{M}}_0$. The reliability of J_0 after executing $\mathrm{PL}_A(J_0)$ is $\widehat{R}_{0,0} = \widehat{\boldsymbol{P}}_0 \boldsymbol{\chi}_0 = \widehat{\boldsymbol{P}}_0[\mathsf{SF}] + \widehat{\boldsymbol{P}}_0[\mathsf{SS}] = 0.7$. Figure 4 shows the states of the MC after the execution of each pull. The

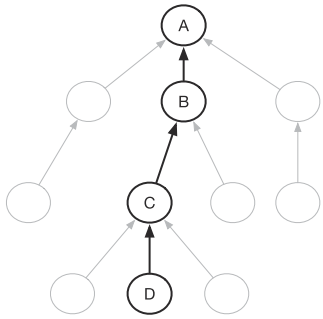


Fig. 5. Multi-hop example.

transition matrices associated with each pull are included at the bottom of the figure. The reliability of flows is evaluated in a similar manner in the remaining slots.

5.3 Synthesizing Recorp Policies for General Topologies

In this section, we extend the results from the previous subsection to general workloads and topologies. Doing so requires that we determine both a coordinator and a service list for each pull. The builder must assign coordinators and service lists such that no transmission and channel conflicts occur. The evaluator must provide lower bounds on the reliability of the flows as they interact across multiple hops. A naive evaluation that simply keeps track of when a coordinator received packets from all combinations of flows does not scale. We will start by discussing how a scalable evaluator may be built and then extend the builder.

5.3.1 The Multi-hop Evaluator. The key insight to building a scalable evaluator is to require coordinator nodes to *operate independently*. Consider a multi-hop flow F_2 shown in Figure 5 whose data is forwarded using the path $\Gamma_2 = \{D, C, B, A\}$. To forward F_2's data, a policy must include a sequence of pulls that have the nodes C, B, and A as coordinators and include F_2 as part of their service lists. A simple approach to ensure that coordinators operate independently is to use an approach similar to the Phase Modification Protocol [5], where a multi-hop flow is divided into single-hop subflows flows and allocate $\delta_2 = D_2/|\Gamma_2|$ slots for the execution of each flow. The first subflow $F_{2,1}$ from D to C is released $\phi_{2,1} = \phi_2$ and must complete with δ_2 slots. The second subflow $F_{2,2}$ from C to B is released at $\phi_{2,2} = \phi_2 + \delta_2$ and it must complete within δ_2 slots. The remainder of the subflows are set up in a similar fashion. To ensure that coordinators operate independently, it is essential that each subflow releases a packet regardless of whether the previous subflow delivered it successfully or unsuccessfully to the next hop. By taking advantage of the independence, we can use the single-hop evaluator described in the previous section to evaluate the reliability of each subflow. Then, the end-to-end reliability of the original flow is simply the product of the reliability of each subflow (due to independence).

The drawback of this approach is that each subflow is allocated an equal number of slots, which can be problematic when the workload of nodes is not uniform. To address this issue, we first convert the end-to-end target reliability of T_i into a local reliability target that each subflow must meet:

$$T_i^{\frac{1}{|\Gamma_i|}}, \tag{4}$$

where $|\Gamma_i|$ is the length of F_i's path measured in hops. Each subflow is then allowed to release the earliest slot in which all subflows associated with the previous hops of the original flow have met

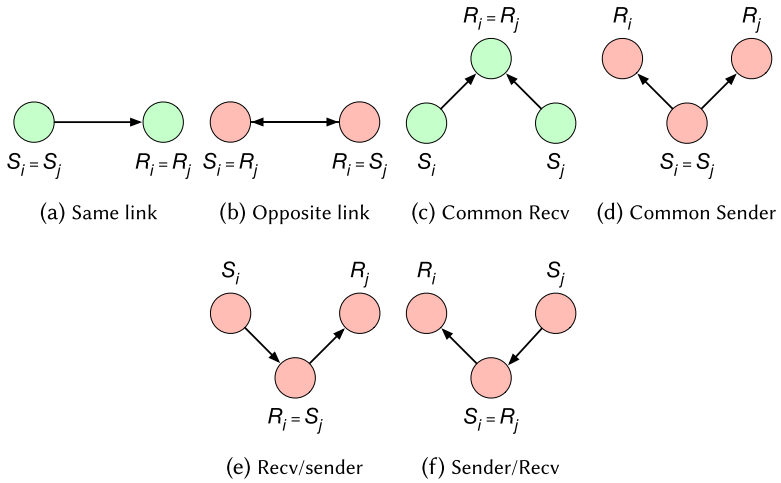


Fig. 6. Possible ways two instances may share at least one node. Green cases have no transmission conflict while red cases do.

the local reliability target. Notably, different subflows may need to be executed a different number of times to meet their local target reliability to handle non-uniform workloads effectively.

5.3.2 The Multi-hop Builder. The optimization problem can be formulated as an **Integer Linear Program (ILP)**. The ILP includes three types of variables. For each node R ($R \in \mathcal{N}$), the variable N_R ($N_R \in \{0, 1\}$) indicates whether R is the coordinator of a pull. For each released instance J_i, the variable I_i ($I_i \in \{0, 1\}$) indicates whether its associated active link will be added to a service list. Finally, variable $C_{R,ch}$ ($C_{R,ch} \in \{0, 1\}$) indicates whether R will use channel ch to communicate. The ILP solution is converted into a set of pulls as follows: for each node R such that $N_R = 1$, we add a pull that has R as the coordinator and a service list with all instances J_i where $I_i = 1$ and R is the receiver of the active link of J_i. The pull is assigned to the entry in the matrix for the current slot and the channel ch for which $C_{R,ch} = 1$. We let $\mathcal{A}$ be the union of the active list of all nodes.

A well-formed policy must ensure that no transmission conflicts will be introduced at run-time. Consider a pull that has R as a coordinator and services instance J_i. Let (SR) be the active link of J_i, where $S = src(J_i)$ and $R = dst(J_i)$. If J_i will be assigned in the current slot (i.e., $I_i = 1$), then S cannot be a coordinator for any other instance, since this would require S to transmit and receive in the same slot. We enforce this using the following constraint:

$$N_S \leq (1 - I_i) \; \forall I_i \in \mathcal{A} : \; S = src(J_i). \tag{5}$$

A similar constraint must also be included for the receiver R. If node R is not a coordinator (i.e., $N_R = 0$), then J_i cannot be assigned and $I_i = 0$. Conversely, if R is selected as a coordinator, instance J_i may (or may not) be assigned (i.e., $I_i \leq N_R = 1$) depending on the objective of the optimization, which we will discuss later in this section. These aspects are captured by the following constraint:

$$I_i \leq N_R \; \forall I_i \in \mathcal{A} : \; R = dst(J_i). \tag{6}$$

The above constraints avoid all transmission conflicts with one exception. Consider the case when two instances J_i and J_j share the same sender but have different receivers. An assignment that respects constraint Equations (5) and (6) is for both instances to be assigned in the current slot (i.e., $I_i = I_j = 1$). However, this would result in a conflict, since the common sender can only

transmit one packet in a slot. To avoid this situation, we introduce the following constraint:

$$I_i + I_j - 1 \leq N_S, \tag{7}$$
$$\forall I_i, I_j \in \mathcal{A} : \; S = src(J_i) = src(J_j) \;\&\; dst(J_i) \neq dst(J_j).$$

Theorem 3. *Constraint Equations (5), (6), and (7) ensure that the execution of pulls will result in no node receiving or transmitting more than once in a time slot.*

Proof. To prove Theorem 3 holds it is sufficient to consider whether two arbitrary flow instances may conflict. Accordingly, there are six cases to be considered, as depicted in Figure 6, where two instances J_i and J_j share at least a node.

Case 1*—Same link (see Figure 6(a)):* If $I_i = I_j = 1$, then $N_{R_i} = N_{R_j} = 1$ due to constraint Equation (6). In this case, both J_i and J_j will be serviced as part of the same Recorp operation that is coordinated by node $R = R_i = R_j$. At run-time, the coordinator R will pull either J_i or J_j (but not both) depending on its local state. Note that this is one the cases Recorp exploits to adapt and improve performance.

Case 2*—Opposite link (see Figure 6(b)):* Executing J_i and J_j in the same slot would result in a conflict, since one of the common nodes would have to be both a sender and a receiver. We will prove by contradiction that J_i and J_j will not be assigned in the same slot. Assume that $I_i = I_j = 1$ and, without loss of generality, that the common node is $N = S_i = R_j$. Since Since $I_j = 1$, then $N_{R_j} = 1$ due to constraint Equation (6). Also, since $I_i = 1$, then $N_{S_i} = 0$ due to constraint Equation (5). This is a contradiction, since $N_{R_j} = N_{S_i}$ and R_j and S_i refer to the same node. The proofs for the cases given in Figures 6(e) and 6(f) are similar.

Case 3*—Common receiver (see Figure 6(c)):* The common receiver case is similar to the same link case with the exception that the senders for both J_i and J_j are different. Note that this is one the cases Recorp exploits to adapt and improve performance.

Case 4*—Common sender (see Figure 6(d)):* Executing J_i and J_j in the same slot would result in a conflict, since $S = S_i = S_j$ would have to transmit two packets in the same slot. We will prove by contradiction that this cannot happen. Assume that $I_i = I_j = 1$. Since $I_i = 1$, then $N_{S_i} = 0$ according to constraint Equation (5). However, $I_i + I_j - 1 = 1 \leq N_{S_i}$ due to constraint Equation (7), which is a contraction. □

The next set of constraints ensures that each pull is assigned a unique channel. We accomplish this by introducing $C_{R,ch}$ to indicate whether coordinator R uses channel ch ($ch = 1 \ldots K$), where K is the number of channels. The selection of channels is subject to the constraints:

$$\sum_{R \in \mathcal{N}} C_{R,ch} \leq 1 \; \forall ch \in 1 \ldots K, \tag{8}$$

$$\sum_{ch=1}^{K} C_{R,ch} = N_R. \tag{9}$$

A requirement of the TLR model described in Section 4 is that coordinators must switch channels between pulls to ensure independence between transmissions. We enforce this property by introducing additional constraints to prevent coordinators from using the same channel.

To enforce the prioritization of instances, we set the optimization objective to be

$$\sum_{i=0}^{i<|\mathcal{A}|} 2^{|\mathcal{A}|-i} I_i. \tag{10}$$

The objective function ensures that a flow F_i will be assigned over lower priority flows unless there is a higher priority flow with a conflict with F_i.

6 EXPERIMENTS

Our experiments demonstrate the efficacy of Recorp to support higher performance and agility than traditional scheduling approaches. We focus on the next generation of smart factories that will use sophisticated sensors that are grid-powered and require higher data rates than current IIoT systems. Specifically, we are interested in answering the following questions:

- Does Recorp improve the real-time capacity in typical IIoT workloads?
- Does Recorp provide safe reliability guarantees as the quality of links varies significantly?
- Can Recorp synthesize policies in a timely manner?

6.1 Methodology

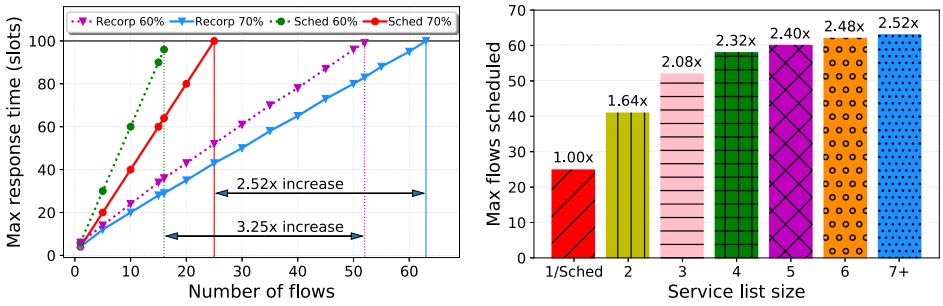
We compare Recorp policies against three baselines. First, we compare two scheduling approaches that do not share entries. To provide a fair comparison between schedules and policies, we first construct **schedules (Sched)** using the same ILP formulation as Recorp policies but without allowing entries to be shared. This is accomplished by adding to the ILP an additional constraint that the size of the service list is one. We also compare against the **conflict aware least laxity first scheduler (CLLF)** [35]. CLLF has been shown to produce near-optimal schedules and constitutes the current state-of-the-art scheduler. Similar to Sched, CLLF also does not share entries. Second, we compare against the **Flow Centric Policy (FCP)** [6], which allows entry sharing only among the links of *a single flow*, whereas Recorp can share entries *across multiple flows*. Sched, CLLF, and FCP utilize sender-initiated transmissions, while Recorp utilizes receiver-initiated pulls.

Unless stated otherwise, we use $m = 70\%$ as suggested by Emerson's guide to deploying WirelessHART networks. In simulations, we set the probability of a successful transmission to equal m. The number of retransmissions used by Recorp, Sched, CLLF, and FCP is configured to achieve a 99% end-to-end reliability for all flows. The period and deadline are equal, and the phases are 0 in all workloads. Flow priorities are assigned such that flows with shorter deadlines have higher priority. To break ties, flows with longer routes are assigned a higher priority. The remaining ties are broken arbitrarily.

We quantify the performance of protocols using *max flows scheduled*, *real-time capacity*, and *response time*. The max flows scheduled measures the maximum number of flows that can be supported without missing the deadlines or reliability requirements of any flows. The real-time capacity is the highest rate at which flows can release packets without missing deadlines or reliability constraints. The response time is the maximum latency of all instances of a flow as measured from the time when an instance is released until it is delivered to its destination.

6.2 Simulations

We use a discrete event simulator to control m in the TLR model precisely, which is impractical on a testbed. The simulator determines the success or failure of transmitting a packet and receiving the acknowledgment over a link by drawing from a Binomial distribution whose change of success can be configured. Unless stated otherwise, all links are configured to have the same success chance of m. All simulations are either single-hop or performed on one of the following two topologies: a 41-node, 6-hop diameter topology with an average of 5.5 links per node derived from a testbed deployed at Washington University in St. Louis (WashU topology) [3] and an 85-node, 6-hop diameter topology with an average of 10.4 links per node derived from the Indriya testbed (Indriya topology) [14]. In simulations, we used settings consistent with 802.15.4: the number of



(a) Max response time and real-time capacity (b) Size Impact of service list on schedulability

Fig. 7. Simulations on star topologies.

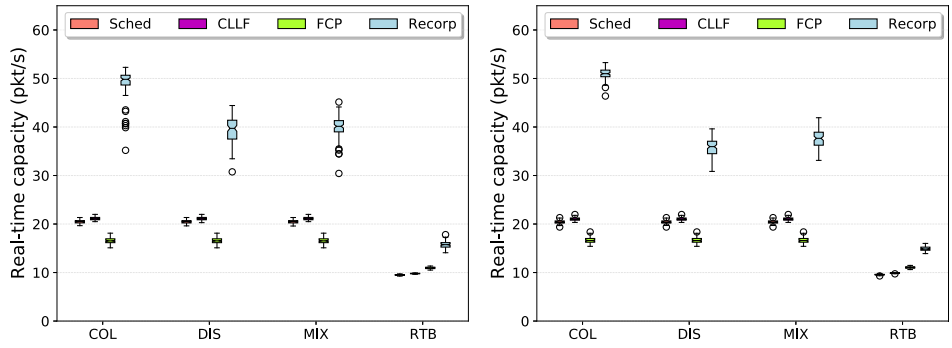
channels was set to 16, and we used 10 ms slots sufficiently large to transmit a packet and receive an acknowledgment.

6.2.1 Star Topology. We compare Recorp and Sched in the practically important case of star topologies. In star topologies, for the workloads we consider, Sched, CLLF, and FCP perform identically and, therefore, we only report the results of Sched. In this experiment, we consider workloads consisting of flows that have a period and deadline of 100 slots. We increase the number of flows until the workload becomes unschedulable under both Recorp and Sched.

Performance in Star Topologies: Figure 7(a) plots the max response time of all scheduled flows as the number of flows in the workload is increased. We configure Sched and Recorp to have an end-to-end reliability of 99% for each flow when $m = 60\%$ and $m = 70\%$. The figure indicates the max response time increased until each protocol reached its real-time capacity, as indicated by the vertical line in the figure. When $m = 70\%$, Recorp supports 63 flows without missing deadlines compared to only 25 flows supported by Sched. This represents a real-time capacity improvement of 2.52 times at $m = 70\%$ and 3.25 times at $m = 60\%$.

Impact of the Service List Size: Schedules and Recorp policies differ in how many instances can share an entry, which can be controlled by constraining the size of the service list. Schedules provide no sharing and are limited to a service list size of one. In contrast, Recorp policies allow multiple flows to be included in the service list to share an entry. Figure 7(b) plots the maximum number of flows scheduled as the service list size is varied when $m = 70\%$. When the size of the service list is one, Recorp behaves like Sched. As we allow more flows to potentially share a slot, the number of flows scheduled increases. However, there are diminishing returns; most of the benefit is observed when the service list is capped at 4 to 6 flows. No meaningful improvement in the real-time capacity may be observed after increasing the service list size beyond 7 flows. Based on this result, we set the maximum service list size to 4 for all remaining experiments. These results indicate that *it is sufficient to share slots across only a few flows to gain most of the benefits of using Recorp policies.*

6.2.2 Multihop Topology. To provide a comprehensive comparison between Recorp, Sched, CLLF, and FCP, we consider four typical workloads: data collection, data dissemination, a mix of data collection and dissemination, and route through the base station. The results presented in this section are obtained from 100 simulation runs for each workload type on each multihop topology. In all runs, the node closest to the center of the target topology is selected as the base station. In each run, we generate 50 flows whose sources and destinations are picked as follows:



(a) 41-node WashU topology (b) 85-node Indriya topology

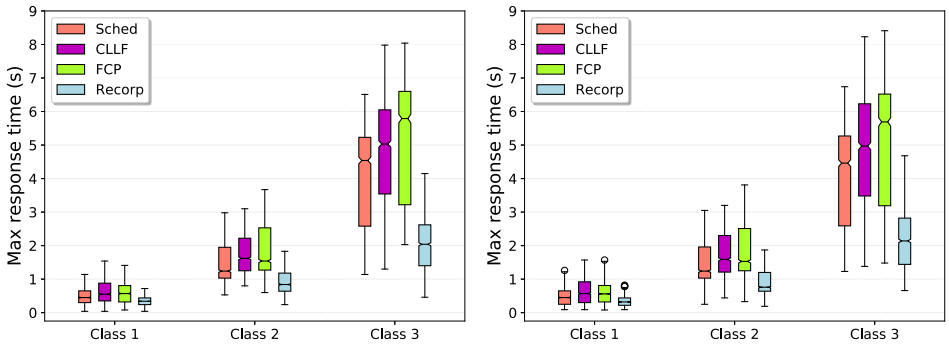
Fig. 8. Real-time capacity results.

- **Data Collection (COL)**: Flows are randomly generated from the nodes to the same base station.
- **Data Dissemination (DIS)**: Flows are randomly generated from the same base station to nodes.
- **Data Collection and Dissemination (MIX)**: Each flow is randomly selected to use either COL or DIS
- **Route Through the Base Station (RTB)**: The source and destination of flows are selected at random, but the routes are constrained to pass through the base station.

Each flow is assigned at random to one of three flow classes whose periods and deadlines maintain a 1:2:5 ratio. For example, if Class 1 has a period of 100 ms, then Class 2 has a period of 200 ms, and Class 3 has a period of 500 ms. We refer to the period of Class 1 as the base period. In a run, the base period of the flows is decreased until the workload is unschedulable. The results of a run are obtained for the smallest base period for which the workload is schedulable.

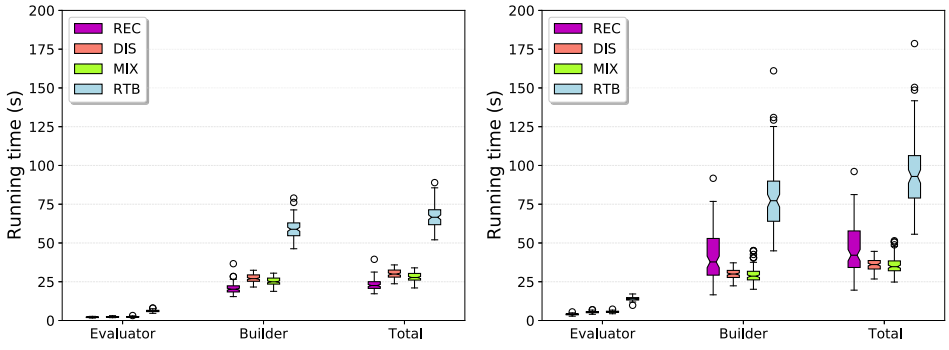
Real-time Capacity and Response Time: Figures 8(a) and 8(b) plot the distribution of the observed real-time capacities for the WashU and Indriya topologies, respectively. FCP provides a median improvement over Sched and CLLF only for the RTB workload. Moreover, the improvement is minor, with FCP increasing real-time capacity by only 1.15 and 1.16 pkt/s over CLLF for the RTB workload in the WashU and Indriya topologies, respectively. The improvement over Sched was similar. For the other workloads where the base station is the source/destination, FCP has worse performance, since sharing within a flow reduces only the utilization of the intermediary nodes on a flow's path, but not on the source and destination nodes. In contrast, Recorp outperforms all other protocols. For example, Recorp outperforms the overall next best protocol, CLLF, by a median margin of 28.74, 18.63, 19.05, and 5.94 pkt/s in the WashU topology and 30.00, 14.96, 16.67, and 4.97 pkt/s in the Indriya topology for the COL, DIS, MIX, and RTB workloads, respectively. Together these results correspond to a median increase in real-time capacity over CLLF of between 50% and 142% across each workload and topology. Moreover, Recorp outperforms both Sched and FCP by similar amounts across all workloads and topologies.

Figures 9(a) and 9(b) show the distribution of the response times of each flow class from the previous experiment for the MIX workload (including all runs). Consistent with the real-time capacity for the MIX workload, FCP underperforms both Sched and CLLF with one exception. For both topologies, FCP provides a slightly lower median response time than CLLF for Class 2. The reason for this, and the reason that CLLF has a higher response time than Sched across all workloads and topologies, is due CLLF making scheduling decisions as a function of remaining conflict-aware



(a) 41-node WashU topology (b) 85-node Indriya topology

Fig. 9. Response time per flow class.



(a) 41-node WashU topology (b) 85-node Indriya topology

Fig. 10. Synthesis time.

laxity. The consequence of this approach is that CLLF occasionally allows lower priority flows to preempt higher priority flows. In contrast, Recorp maintains deadline-monotonic prioritization and reduces the response time for all classes in both topologies, with particularly good performance for the middle and lowest priority flow classes. Specifically, Recorp decreased the median response time in the WashU topology by 0.11 s, 0.40 s, and 2.50 s and in the Indriya topology by 0.13 s, 0.48 s, and 2.32 s over the next best protocol, Sched, for flow Class 1, Class 2, and Class 3, respectively. Similar trends and performance differences were observed for the other workloads under all topologies, with one exception. FCP slightly outperformed Sched in the RTB workload across flow classes and topologies. However, Recorp still significantly outperformed Sched, CLLF, and FCP. These results indicate *Recorp policies can significantly improve real-time capacity and response times for common IIoT workloads.*

Synthesis Time: Next, we turn our attention to the feasibility of synthesizing policies. Typical IIoT systems have workloads that are stable for tens of minutes, which justifies synthesizing Recorp policies. We divided the total time to synthesize a policy into two categories: the time the evaluator spends managing the system state and the time the builder spends solving ILPs to determine the pulls in each slot. Figures 10(a) and 10(b) plot the distribution of the execution times for each workload under the WashU and Indriya topologies, respectively. The median total synthesis time is below 93 s for all workloads and both topologies. The synthesis time of the route through the

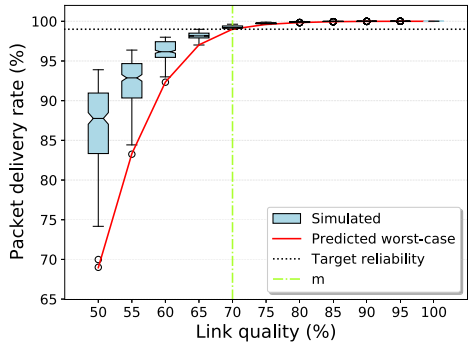


Fig. 11. PDR at different link qualities for a representative MIX workload in the WashU topology for $m = 70\%$.

base station is significantly higher than the other workloads, as flows tend to have longer paths. This results in more states to be managed and longer schedules. The builder tends to be the most expensive, followed by the evaluator. We plan to explore ways to reduce the synthesis time further. These results indicate that *it is feasible to synthesize policies within 1–3 min for realistic networks.*

Threshold Link Reliability Model Evaluation: Next, consider Recorp's reliability guarantees. Recorp uses a safe lower bound on a flow's end-to-end reliability under the TLR model (i.e., when the link quality of a flow exceeds m) described in Theorem 2. We are interested in providing simulated and empirical evidence that the lower bound is safe. Additionally, when the link quality degrades below m, Recorp provides no performance guarantees. However, the end-to-end reliability of flows should degrade gracefully as link quality falls below m.

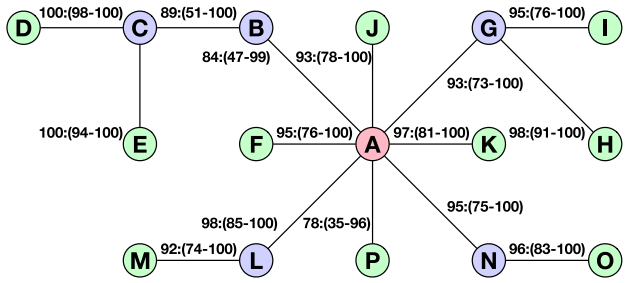
To this end, we simulated a representative MIX workload Recorp policy on the WashU topology with $m = 70\%$. We varied the link quality from 50% to 100% in increments of 5%. For each setting, we simulated 1,000,000 hyperperiods and recorded each flow instance's outcome in each hyperperiod, delivering their data successfully or otherwise. For each instance, we computed the probability of delivering its data and plotted the distribution of all instances as "Simulated" in Figure 11. Additionally, we used the evaluator to compute the lower bound on each instance's reliability. We plotted the worst-case reliability across all instances in the same figure as "Predicted worst case." The worst-case bounds computed by the evaluator are smaller than those predicted through simulations for all test link qualities indicating that they are safe (i.e., Theorem 2 holds). As expected, when link quality exceeds $m = 70\%$, all instances had reliability above their target end-to-end reliability of 99%. When the link quality is below $m = 70\%$, Recorp provides no guarantees regarding the reliability of flows. Nevertheless, the results indicate that the reliability of flows degrades gracefully as link quality deteriorates. In the next section, we additionally validate the safety of the TLR model on a real testbed.

6.3 Testbed Results

We evaluated Recorp and the baselines on a testbed of 16 TelosB motes deployed at the Univesity of Iowa (see Figure 12(b)). At the beginning of each protocol's hyperperiod 3 slots are reserved for a broadcast graph that is used to control traffic and time synchronization. When a parent broadcasts a packet, it includes its current time in the packet. The children detect the start-of-frame delimiter upon receiving the packet and adjust their clocks to match their parent's clock. We consider a data collection workload that involves ten flows with equal periods whose routes are included in Figure 12(a). We configured Recorp, Sched, and FCP to provide an end-to-end reliability of 99% when $m = 70\%$. We did not consider CLLF in this experiment, since CLLF provided nearly identical

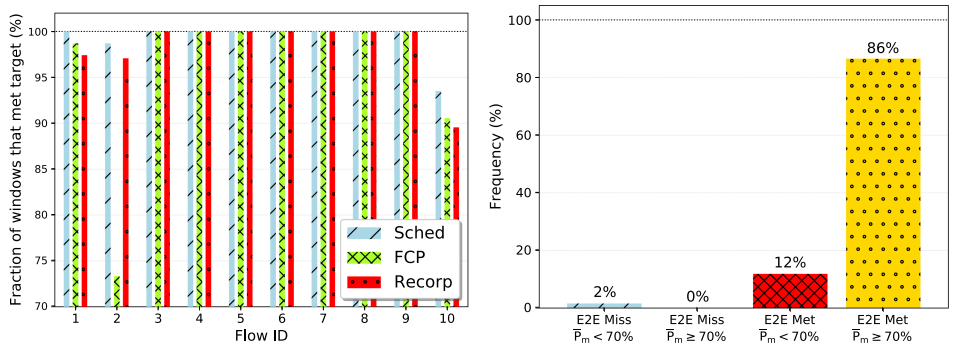
Flow	Route
1	D→C→B→A
2	E→C→B→A
3	H→G→A
4	I→G→A
5	M→L→A
6	O→N→A
7	F→A
8	J→A
9	K→A
10	P→A

(a) Flows



(b) Topology

Fig. 12. Testbed topology and flow routes. The green, purple, and red nodes indicate the flow sources, intermediary nodes, and the base station, respectively. Link quality (with interference) was calculated over a sliding interval of 100 runs (about 200 s). The min, median, and max observed link quality over all intervals for each link is given as median:(min-max).



(a) End-to-end flow reliability

(b) Safety of reliability bound

Fig. 13. Evaluating the safety of the reliability bound.

performance to Sched. The experiments use 802.15.4 channels 11, 12, 13, and 14, which overlap with the 802.11g WiFi network co-located in the building. We have evaluated the performance of Recorp with and without additional interference generated by a laptop near the base station, which transmitted ping packets at a rate of 1.5 Mbps. When no interference was present, all flows met their end-to-end reliability, and the quality of the links exceeded $m = 70\%$. In the following, we will focus on when interference was present to evaluate Recorp's ability to adapt in an environment with significant link quality variation. We organized our experiments into multiple runs, each run consisting of running the schedule/policy of each protocol for one hyperperiod and storing the outcome of each transmission to flash at the end of the run. The reported results were obtained from releasing 10,000 packets for each protocol (i.e., 10,000 runs) over approximately 6 h.

Real-time Capacity and Reliability: We determined the maximum rates of the ten data collection flows that can be supported using Recorp, Sched, and FCP. Recorp provides a real-time capacity of 38.46 pkt/s compared to 19.6 and 18.2 pkt/s provided by Sched and FCP, respectively. The real-time capacity of Recorp is 96% higher than Sched. This result is consistent with the multihop experiments where Recorp significantly outperforms the baselines. Next, we will evaluate whether the improved capacity comes at the cost of lower reliability.

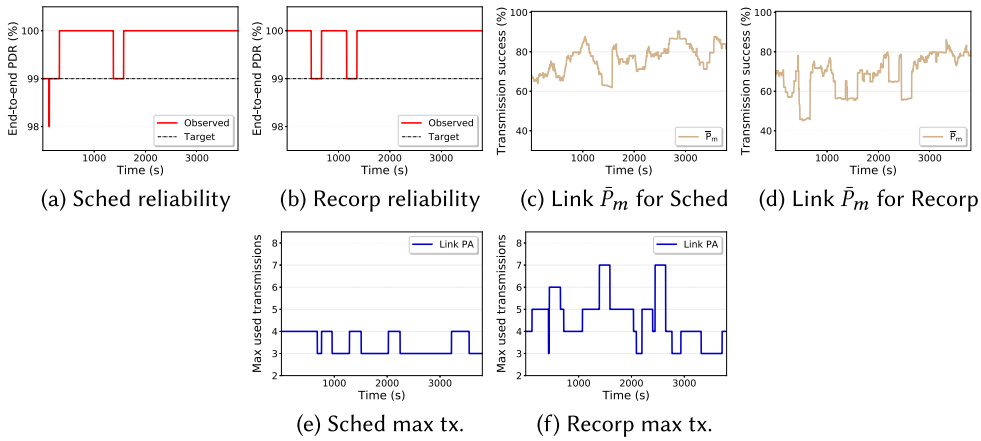
We computed the **packet delivery rate (PDR)** over sliding windows of 100 runs. In Figure 13(a), we plot the fraction of windows that met the end-to-end reliability target of 99% for each protocol. The lowest reliability was observed for FCP's flow 2. We found that the root cause behind the lower performance of FCP is the contention-based mechanism used to arbitrate access to the entries shared by the links of a flow. FCP prioritizes the transmission of nodes closer to the flow's destination by having them transmit at the beginning of the slot while the other nodes only transmit after **clear channel assessment (CCA)** indicates the slot is not used. In the presence of WiFi interference, CCA was not a robust indicator of transmissions. This experience highlights the potential advantage of using receiver-initiated pulls over contention-based approaches that rely on CCA.

Recorp policies guarantee probabilistically that the end-to-end reliability constraints are met as long as the quality of all used links exceeds a minimum packet reception rate m. When the quality of the links falls below m, we provide no guarantees on the end-to-end reliability of flows. We evaluate whether our guarantee holds as follows. Based on the trace of successes and failures observed during the experiment, we fit a Bernoulli $\bar{P}_m$ random variable to lower bound the observed failure distributions. Accordingly, Recorp's analytical bounds on flow reliability hold only if $\bar{P}_m \geq P_m = 70\%$. Figure 13(b) classifies each window of 100 runs into the following cases:

(1) Case $\bar{P}_m \geq 70\%$, E2E Met: For 86% of the windows, the minimum link quality met or exceeded 70% (i.e., $70\% = m \leq \bar{P}_m$). Over all these windows, Recorp policies indeed guaranteed that the end-to-end reliability of all flows exceeded the 99% target.
(2) Case $\bar{P}_m \geq 70\%$, E2E Miss: There are no cases where the minimum link quality exceeds 70%, and the flows do not meet the target 99% reliability. These first two cases demonstrate that the TLR model is safe, since no flows miss their end-to-end reliability targets when the minimum link quality is met.
(3) Case $\bar{P}_m < 70\%$: When the actual link quality falls below the minimum link quality of $m = 70\%$, we provide no guarantees on the flow's reliability. Out of the 14% of windows where $\bar{P}_m < 70\%$, in 12%, the end-to-end reliability is met, while for the other 2%, it is not.

These experiments show that *Recorp policies can significantly improve real-time capacity while meeting the end-to-end reliability of flows as the quality of links fluctuates above the minimum link quality m.*

Effective Adaptation: To analyze Recorp's ability to adapt to variations in link quality, we consider the trace of Sched and Recorp for flow 10, which exhibits the lowest link reliability and highest variability in our experiments. Figure 14 plots the end-to-end reliability (after retransmissions), the parameter $\bar{P}_m$ of a Bernoulli distribution that is fitted to account for the burst of failures observed empirically in each window, and the maximum number transmissions used by Sched and Recorp over a trace of 4,000 s. Notably, the end-to-end reliability of Sched and Recorp is similar during this time frame (Figures 14(a) and 14(b)). Recorp achieves a similar level of end-to-end reliability by performing more retransmissions, as it is clear from comparing Figures 14(e) and 14(f). Sched uses 3–4 maximum retransmissions over the course of the hour but notably still briefly missed the end-to-end PDR target. In contrast, Recorp uses between 3 and 7 retransmissions to combat a slightly lower link quality it experienced and did not miss the end-to-end PDR target over the interval. Remarkably, Recorp can (almost) double the number of retransmissions that may be used for flow 10 over Sched without degrading the performance of other flows. These results indicate *that Recorp can provide higher agility than schedules by using its lightweight and local run-time adaptation mechanism to reallocate retransmissions in response to variations in link quality.*



(a) Sched reliability (b) Recorp reliability (c) Link $\bar{P}_m$ for Sched (d) Link $\bar{P}_m$ for Recorp

(e) Sched max tx. (f) Recorp max tx.

Fig. 14. Performance of Recorp and Sched on flow 10 over time.

7 DISCUSSION

7.1 Deployment

Wireless networks that support IIoT applications require careful planning and deployment. The deployment process usually involves profiling the quality of links and the interference on all 16 channels. The collected statistics are used to ensure that there are redundant routes that connect each node to the base station whose link quality exceeds the m threshold of the TLR model. Consistent with Emerson's guide for deploying WirelessHART networks, the value of m is usually set to 60%–70%. Additionally, the channels that have consistently poor reliability are usually blacklisted [22].

7.2 Handling Network Dynamics

Recorp's design focuses on supporting the communication needs of IIoT applications with long-running real-time flows. The network manager uses the current set of flows to build a policy that meets a flow's end-to-end reliability as long as the link quality exceeds m. This approach makes it feasible to run the same policy for prolonged periods of time without modification. However, the industrial environment may change and lead to node and link failures. The primary mechanism used by Recorp to adapt to topology changes and node failures is to synthesize new policies. However, the frequency with which new policies need to be synthesized can be reduced by integrating Recorp with multi-hop routing techniques (e.g., References [11, 34]) to allow Recorp to tolerate some link or node failures without having to reconstruct policies. In the following, we describe a centralized management and control plane that can detect and adapt to node failures and topology changes using an approach similar to WirelessHART.

The network manager uses three different types of specialized flows to implement the control plane. These specialized flows are periodic, but unlike regular flows, they have a higher priority then regular and require different mechanisms to allocate their slots. A *dissemination flow* is used to disseminate policies to all nodes after its synthesis. A dissemination flow reserves a single slot during which the base station floods a packet to all the nodes. An efficient approach to handle this type of communication is to use GLOSSY floods [18]. A *join flow* reserves a single slot during which all nodes wait to listen to a fixed channel for nodes to request joining the network. A node wanting to join will use CSMA techniques to broadcasts its request to join the network. All nodes receiving the request will forward it to the base station using a *report flow*. A *report flow* is used to

inform the network manager about the status of nodes and links. A report flow is set up from each leaf node in the upstream graph to the base station. As the packet of a report flow is forwarded, nodes along the paths may append to the payload node and link health information to be delivered to the network manager. Each node along the path is provided a fixed number of bytes that they may use. As described next, node and link failure reports are prioritized while the remaining space is used for the quality of links that are not currently part of the routing tree. Node and link health reports can also be piggy-backed onto periodic traffic to improve network agility.

Each node collects statistics about the bursts of packet losses within a window of slots to assess the quality of links currently in use. As described in Section 6.3, a node uses this information to fit a Binomial distribution whose chance of success P_s is sufficiently large to account for the observed burst. On the one hand, if $P_s < m$, then the TLR assumptions are violated, and the network manager is notified immediately about the link failure. Accordingly, this information is included in the report flow. On the other hand, if $P_s \geq m$, then the TLR assumptions are not violated. This information is not as urgent to the network's operation and is included in a control packet only if there is room available. This approach can also be extended to collect statistics of links that are currently not in use. However, a node should only use the reception of the data packets (ignoring the pull requests) to estimate the one-way link quality between itself and the packet's sender. This information should be included infrequently in control packets to allow the network manager to update the upstream and downstream graphs.

Recorp allows nodes to join or leave the network dynamically. A node wanting to join the network first listens to a fixed channel until it receives the packet of a report flow that allows it to synchronize with the network and learn the join flows' parameters. When the join flow is released, the node broadcasts a request to join the network, which is routed to the network manager using the next available report flow. Upon receiving a join request, the network manager updates the upstream and downstream graphs to include the new node and starts the synthesis of a new policy. When the synthesis is complete, the policy is disseminated to all nodes, including the new node. A node leaving the network uses a report flow to send its request to the base station.

7.3 Handling Other Traffic

Recorp is optimized for improving the performance of real-time flows that are expected to carry the bulk of the traffic in IIoT applications. However, other types of traffic may also exist. For example, IIoT applications may benefit from supporting even-triggered emergency communication in response to unsafe situations or failures. Recorp can support emergency communication using techniques proposed in Reference [26]: each slot is modified such that emergency traffic is transmitted at the beginning of the slot. In contrast, regular traffic is transmitted after a short delay. Other examples of traffic include aperiodic communication. The simplest solution for handling these transmissions is to dedicate slots for their transmission periodically. Transmissions during these slots are done using typical CSMA/CA techniques. This approach reserves a portion of the bandwidth for other types of traffic. Moreover, it is straightforward to account for these additional slots in our analysis.

7.4 High Data-Rates and Energy Efficiency

Recorp is designed for IIoT applications that require high data rates and usually use grid-powered nodes. Our simulation and testbed results are performed using the IEEE 802.15.4 physical layer. The standard supports a maximum packet size of 128 bytes and a maximum data rate of 250 kbps. Under these settings, the maximum real-time capacity from the testbed experiments equates to only approximately 39 Kbps. While this data rate may be able to meet the real-time and reliability requirements of some high data rate sensors such as torque and temperature sensors with update

rates on the order of 10 – 500 ms [43], it is unlikely to be sufficient for microphones and cameras. For this type of application, Recorp can be used unmodified with IEEE 802.15.4a UWB. IEEE 802.15.4a provides a significantly higher data rate of 27.24 Mbps and some UWB radios (e.g., DWM1001) support packets as large as 1024 bytes. In future work, we will explore using other physical layers to extend Recorp's applicability further.

One of the limitations of Recorp is its potentially high energy usage. Indeed, an entry in the scheduling matrix using the Sched protocol will involve at most two nodes using their radios. In contrast, a Recorp pull may involve as many as five nodes for a service list of size four. The nodes in the service list size must turn on their radio for a short duration to determine whether the coordinator will request information from them. As a result, the real-time capacity and response time improvements offered by Recorp come with the cost of additional energy consumption. However, industrial applications that require higher data rates usually use grid-powered sensors. Additionally, many applications use a powered backbone to carry high data-rate traffic while including some battery-powered nodes (e.g., References [4, 7, 10]). Recorp can be configured in such scenarios to use service lists size of size one on battery-operated devices to achieve the same energy consumption level as existing scheduling approaches. Moreover, Recorp could use larger service list sizes for the powered nodes to provide higher throughput and lower latency.

8 RELATED WORK

Due to its predictability, TDMA has become the de facto standard for IIoT systems. There are many scheduling algorithms to construct TDMA schedules (e.g., References [16, 31, 35, 37, 39]). However, a common weakness of TDMA protocols is their lack of adaptability to network dynamics. To address this limitation, various techniques to handle variations in link quality, topology changes, and fluctuations in workloads have been proposed (e.g., References [13, 20, 30]). In this article, we focus on handling variations in link quality, as they are common in harsh industrial environments [8, 17]. Our work is complementary to and may be integrated with techniques designed to handle other types of network dynamics.

Researchers have considered various approaches to combining CSMA and TDMA into hybrid protocols, ultimately sacrificing either flexibility or predictability. A common approach to combine CSMA and TDMA is to have each protocol run in different slots. This approach is adopted in industrial standards such as WirelessHART [2] and ISA100.11a [1]. However, predictable performance cannot be provided for the traffic carried in CSMA slots. Another alternative is to dynamically reuse slots (e.g., Reference [33]) or transmit high-priority traffic (e.g., Reference [27]) by selecting primary and secondary slot owners. In this approach, slot owners are given preference to transmit and send data using a short initial back-off. If a slot owner does not have any data to transmit, then other nodes may contend for its use after some additional delay. A generalization of this scheme is prioritized MACs that divide a slot into sub-slots to provide different levels of priority [36]. However, none of these protocols provide analytical bounds on their performance. In contrast to the above approaches that involve carrier sensing, our policies rely on receiver-initiated polling and the local state of nodes to adapt. We expect policies to be less brittle in practice than solutions that use carrier sense as they do not require tight time synchronization for adaptation.

Several distributed protocols for constructing TSCH schedules that support best-effort [15, 40] and real-time [42] traffic have been proposed. Our work is complementary, since these works focus primarily on handling workload changes while we focus on adapting to variations in link quality over short time scales. These protocols can't adapt at the time scales required to handle link quality variations due to their communication overheads. Our approach combines offline policy synthesis with local adaptation performed at run-time. This approach can effectively handle changes over short time scales as the adaptation process is local and lightweight.

Transient link failures are common in wireless networks [9, 38] and even more prevalent in harsh industrial environments [8, 17]. The state-of-the-art is to schedule a fixed number of retransmissions for each link, potentially using different channels. Little consideration is usually given to selecting the correct number of retransmissions based on link quality. Recently, some work has been done to tune the number of retransmissions based on the burstiness of links [30, 41]. While this is a step in the right direction, the fundamental problem is that links are treated in isolation and provisioned to handle worst-case behavior in a fixed manner. As a result, retransmissions cannot be redistributed across links as needed at run-time. A notable exception is our prior work [6], which proposes a technique to share transmissions among the links of a flow at run-time. However, this technique's performance benefits are sensitive to the length of flows, with the most benefit occurring in large multi-hop networks uncommon in practice. Our experiments show that this approach is only effective when flows are routed through the base station and not for the more common data collection and dissemination scenarios. By enabling entries to be *shared across flows*, we can significantly reduce the number of slots needed by flows to meet their end-to-end reliability, resulting in significant performance improvements.

9 CONCLUSIONS

Recorp is a practical and effective solution for IIoT applications that require predictable, real-time, and reliable communication in dynamic wireless environments. We leverage the stability of IIoT workloads and the improving resources of wireless nodes to build a solution that combines offline policy construction and run-time adaptation. A Recorp policy assigns a Recorp operation to each slot and channel, which specifies a coordinator that will arbitrate channel access and a list of flows that may be serviced. At run-time, the coordinator dynamically executes the flows in the service list from which it has not received a packet. The advantage of Recorp is that nodes can locally reallocate the retransmissions of flows in response to variations in link quality and, as a result, provide higher performance than scheduling approaches.

The synthesis of policies required us to address two key challenges: handling the state explosion problem and providing predictable performance as the quality of links varies. We developed a practical approach to synthesize policies iteratively. In each slot, the builder employs an ILP program to determine the Recorp operations that will be performed in the current slot. Based on the selected operations, the evaluator determines a lower-bound on the end-to-end reliability of each flow to determine if it met its target end-to-end reliability. A key advantage of Recorp is that it provides guarantees when slots are shared under a realistic model of wireless communication. Specifically, we guarantee that a constructed Recorp policy will meet a user-specified reliability and deadline constraint for each flow as long as the quality of all (used) links exceeds a minimum link quality.

We have extensively evaluated the performance of Recorp through both simulations and testbed experiments. Our results indicate that due to their increased agility, Recorp policies can significantly improve real-time capacity (median 50%–142%) and reduce worst-case response time (median 27%–70%) while meeting a specified end-to-end reliability. These trends hold across typical IIoT workloads, including data collection, data dissemination, and route through the base station. Additionally, we showed empirically that our theoretical guarantees of real-time performance and reliability hold even in the presence of significant interference.

APPENDIX

A PROOF OF THEOREM 2

In this section, we prove Theorem 2. Before proving the theorem though, we will introduce some definitions and lemmas. We will illustrate their use using a single-hop scenario with two flows F_0

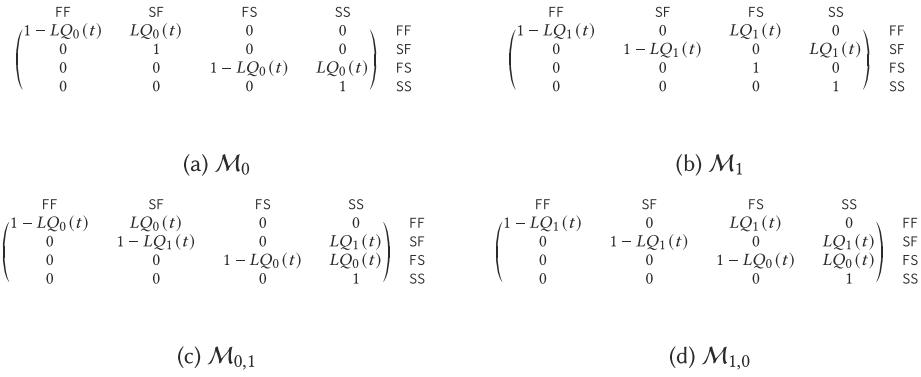


Fig. 15. Possible transition matrices when two flows are active.

and F_1 ($\mathcal{F} = \{F_0, F_1\}$) that relay data to the base station (see Figure 3(a)). In the following, we let $N = |\mathcal{F}|$. We consider the execution of two generic instances—J_0 and J_1—of these flows.

Under the considered example, the state of the system is represented as a vector where the ith entry indicates whether the currently released instance of flow i was received successfully (S) or not (F) by the base station. Accordingly, the states of our example are FF, SF, FS, and SS. There are four possible pulls that may be performed in a slot t: $\text{PL}_A(J_0)$, $\text{PL}_A(J_1)$, $\text{PL}_A(J_0, J_1)$, and $\text{PL}_A(J_1, J_0)$. Note that the builder described in Section 5.2 would never assign $\text{PL}_A(J_1, J_0)$ as it strictly enforces prioritization among flows. Nevertheless, the theorem and lemmas presented in this section apply to a broader class of builders that allow priority inversions and may assign $\text{PL}_A(J_1, J_0)$. For each pull, we construct an associated transition matrix according to Algorithm 1:

- $\mathcal{M}_0$—the transition matrix associated with $\text{PL}_A(J_0)$,
- $\mathcal{M}_1$—the transition matrix associated with $\text{PL}_A(J_1)$,
- $\mathcal{M}_{0,1}$—the transition matrix associated with $\text{PL}_A(J_0, J_1)$,
- $\mathcal{M}_{1,0}$—the transition matrix associated with $\text{PL}_A(J_1, J_0)$.

Each of the matrices for the considered example are included in Figure 15. Note that all of the transition matrices depend on the quality of the links $LQ_0(t)$ and $LQ_1(t)$ at time t.

According to Equation (1), the network state after executing t pulls is

$$\boldsymbol{P}_t = s_0^T \mathcal{M}_{srv(0)} \mathcal{M}_{srv(1)} \cdots \mathcal{M}_{srv(t)},$$

where s_0 is an initial state and $\mathcal{M}_{srv(t')}$ is the transition matrix associated with the pull performed in slot t', $0 \leq t' \leq t$, and in our example is equal to either $\mathcal{M}_0$, $\mathcal{M}_1$, $\mathcal{M}_{0,1}$ or $\mathcal{M}_{1,0}$. This equation describes the state evolution of a MC over time. Note that unlike traditional MCs, the transition matrix of this MC is parametric and the value of those parameters change over time.

The transition matrices have a special structure, which we will characterize next. We impose a partial order on the states that reflects how the network changes its state in response to a successful pulls (see procedure onSuccess() of Algorithm 1).

Definition A.1. We say the states s_1 and s_2 are partially ordered, $s_1 \preceq s_2$, if and only if the following is true:

$$s_1[k] = \text{S} \Rightarrow s_2[k] = \text{S} \; \forall k \in [0, N).$$

The partial order induced by $\preceq$ in our example is: FF $\preceq$ SF $\preceq$ SS and FF $\preceq$ FS $\preceq$ SS. The states SF and FS are not comparable. Relating $\preceq$ to the onSuccess() method, the ordering FF $\preceq$ SF implies that there is a service list srv (e.g., $srv = \{J_0\}$ or $srv = \{J_0, J_1\}$) such that onSuccess(FF, J_0) = SF. We make two observations of this partial order:

Lemma A.2. $s_1 \preceq \text{onSuccess}(s_1, J_k)$ *for all instances* J_k.

Proof. onSuccess can change only the kth entry in s_1 to S. If $s_1[k] = \mathsf{S}$, then the partial order holds as the state will not change (i.e. $s_1 = \text{onSuccess}(s_1, J_k)$). If $s_1[k] = \mathsf{F}$, then the kth entry in s_1 will change to S and all other entries will stay the same. This also does not violate the partial order. □

Lemma A.3. *If* $s_1 \preceq s_2$, *then* $\text{onSuccess}(s_1, J_k) \preceq \text{onSuccess}(s_2, J_k)$, *for all instances* J_k.

Proof. onSuccess can change only the kth entry of a state so there are four possibilities. (1) If $s_1[k] = \mathsf{S}$ and $s_2[k] = \mathsf{S}$, then $s_1 = \text{onSuccess}(s_1, J_k)$ and $s_2 = \text{onSuccess}(s_2, J_k)$. Therefore, $\text{onSuccess}(s_1, J_k) \preceq \text{onSuccess}(s_2, J_k)$. (2) If $s_1[k] = \mathsf{F}$ and $s_2[k] = \mathsf{F}$, then the kth entry of s_1 and s_2 will change to S and all other entries will stay the same. Therefore, $\text{onSuccess}(s_1, J_k) \preceq \text{onSuccess}(s_2, J_k)$. (3) If $s_1[k] = \mathsf{S}$ and $s_2[k] = \mathsf{F}$, then the assumed partial ordering is violated and therefore the lemma is not violated. (4) If $s_1[k] = \mathsf{F}$ and $s_2[k] = \mathsf{S}$, then the kth entry of s_1 will change to S with all other entries staying the same and $s_2 = \text{onSuccess}(s_2, J_k)$. Since $s_2[k] = \mathsf{S}$, $\text{onSuccess}(s_1, J_k) \preceq \text{onSuccess}(s_2, J_k)$. □

We will use the notation $\mathcal{M}_{srv(t)}[i, j]$ to refer to the i, j element of the matrix and $\mathcal{M}_{srv(t)}[i, :]$ to refer to the ith row. The values of $\mathcal{M}_{srv(t)}[i, :]$ include the likelihood of transitioning from s_i to another state in Ψ. The values of a row follow one of two patterns: (1) If the current state is s_i, J_k is an instance in the current service list to be executed such that $s_i[k] = \mathsf{F}$, and $s_j = \text{onSuccess}(s_i, J_k)$, then all entries in $\mathcal{M}_{srv(t)}[i, :]$ are zero except for $\mathcal{M}_{srv(t)}[i, i] = 1 - LQ_k(t)$ and $\mathcal{M}_{srv(t)}[i, j] = LQ_k(t)$. (2) Otherwise, if the current state is s_i, then there is only one non-zero entry in $\mathcal{M}_{srv(t)}[i, :]$ and it is $\mathcal{M}_{srv(t)}[i, i] = 1$. Based on these observations, we can rewrite $\mathcal{M}_{srv(t)}$ as

$$\mathcal{M}_{srv(t)} = \boldsymbol{I} + LQ_0(t)E_0 + LQ_1(t)E_1 + \cdots + LQ_N(t)E_N = \boldsymbol{I} + \sum_{i=0}^{N} LQ_i(t)E_i, \tag{11}$$

where $\boldsymbol{I}$ is the identity matrix and matrix $E_i(t)$ has the following properties: (1) $E_i(t)$ is upper-triangular, (2) the entries of $E_i(t)$ are in $\{-1, 0, 1\}$, and (3) in each row, $E_i(t)[i, :]$, there is either exactly one +1 entry off the diagonal and one −1 entry on the diagonal or all the entries of the row are zero. As an example, the transition matrix $\mathcal{M}_{0,1}$ may be rewritten as

$$\begin{aligned}\mathcal{M}_{0,1} &= \boldsymbol{I} + LQ_0(t)E_0(t) + LQ_1(t)E_1(t)\\ &= \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix} + LQ_0(t)\begin{pmatrix}-1&1&0&0\\0&0&0&0\\0&0&-1&1\\0&0&0&0\end{pmatrix} + LQ_1(t)\begin{pmatrix}0&0&0&0\\0&-1&0&1\\0&0&0&0\\0&0&0&0\end{pmatrix}.\end{aligned}$$

We now create the following definition to relate the partial ordering to the actual state probabilities and make the following two observations.

Definition A.4. A vector $\boldsymbol{f}$ given the partial order induced by $\preceq$, if $s_i \preceq s_j$ implies $\boldsymbol{f}[i] \leq \boldsymbol{f}[j]$.

Lemma A.5. *If* $\boldsymbol{f}^T$ *is an increasing vector and* $\mathcal{M}_{srv(t)}$ *is a transition matrix, then* $\boldsymbol{g}^T = \boldsymbol{f}^T \mathcal{M}^T_{srv(t)}$ *is also an increasing vector.*

Proof. Consider an arbitrary instance J_k and let $s_i \preceq s_j$, $s_a = \text{onSuccess}(s_i, J_k)$, and $s_b = \text{onSuccess}(s_j, J_k)$. Consider the ith and jth entries of $\boldsymbol{g}^T$:

$$\begin{aligned}\boldsymbol{g}^T[i] &= \boldsymbol{f}^T[i](1 - LQ_k(t)) + \boldsymbol{f}^T[a]LQ_k(t),\\ \boldsymbol{g}^T[j] &= \boldsymbol{f}^T[j](1 - LQ_k(t)) + \boldsymbol{f}^T[b]LQ_k(t).\end{aligned}$$

Notice that $\boldsymbol{f}^T[i] \leq \boldsymbol{f}^T[j]$ by definition, since $s_i \preceq s_j$ and $\boldsymbol{f}^T[a] \leq \boldsymbol{f}^T[b]$ by Lemma A.3. As a result, we can conclude $\boldsymbol{g}^T[i] \leq \boldsymbol{g}^T[j]$. Since $\boldsymbol{g}^T[i] \leq \boldsymbol{g}^T[j]$ holds for an arbitrary instance J_k, $\boldsymbol{g}^T$ must be an increasing vector. □

Lemma A.6. *If $\boldsymbol{f}^T$ is an increasing vector, $\boldsymbol{g}^T = \boldsymbol{f}^T \mathcal{M}^T_{srv(t)}$, and $\boldsymbol{g'}^T = \boldsymbol{f}^T \widehat{\mathcal{M}}^T_{srv(t)}$ with $\widehat{\mathcal{M}}_{srv(t)} = (\boldsymbol{I} + \sum_{i=0}^{N} mE_i(t))$ and $LQ_i(t) \geq m$, then $\boldsymbol{g}^T \geq \boldsymbol{g'}^T$ component-wise.*

Proof. Consider $\boldsymbol{g}^T - \boldsymbol{g'}^T$:

$$\begin{aligned}\boldsymbol{g}^T - \boldsymbol{g'}^T &= \boldsymbol{f}^T \mathcal{M}^T_{srv(t)} - \boldsymbol{f}^T \widehat{\mathcal{M}}^T_{srv(t)} \\ &= \boldsymbol{f}^T \left(\boldsymbol{I} + \sum_{i=0}^{N} LQ_i(t)E_i(t)\right)^T - \boldsymbol{f}^T \left(\boldsymbol{I} + \sum_{i=0}^{N} mE_i(t)\right)^T \\ &= \left(\sum_{i=0}^{N} (LQ_i(t) - m)E_i(t)\right)\boldsymbol{f}.\end{aligned}$$

Consider now an arbitrary instance J_k and state s_i such that $s_a = \text{onSuccess}(s_i, J_k)$. By Lemma A.2, $s_i \preceq s_a$. Since $\boldsymbol{f}$ is an increasing vector (because $\boldsymbol{f}^T$ is an increasing vector), $\boldsymbol{f}[i] \leq \boldsymbol{f}[a] \implies 0 \leq \boldsymbol{f}[a] - \boldsymbol{f}[i]$. Notice that either $E_i(t)[i,i] = E_i(t)[i,a] = 0$ or $E_i(t)[i,i] = -1$ and $E_i(t)[i,a] = 1$. If $E_i(t)[i,i] = E_i(t)[i,a] = 0$, then

$$\left(\sum_{i=0}^{N} (LQ_i(t) - m)E_i(t)\right)[i,:]\boldsymbol{f} = 0.$$

If instead $E_i(t)[i,i] = -1$ and $E_i(t)[i,a] = 1$, then

$$\begin{aligned}\left(\sum_{i=0}^{N} (LQ_i(t) - m)E_i(t)\right)[i,:]\boldsymbol{f} &= (LQ_i(t) - m)\boldsymbol{f}[a] - (LQ_i(t) - m)\boldsymbol{f}[i] \\ &\geq 0.\end{aligned}$$

Since this result holds for an arbitrary instance J_k, $\boldsymbol{g}^T \geq \boldsymbol{g'}^T$ component-wise. □

We are now prepared to prove Theorem 2, which we reproduce below for convenience.

Theorem 2. *Consider a star topology that has node A as a base station and a set of flows $\mathcal{F} = \{F_0, F_1, \ldots F_N\}$ that have A as destination. Let $LQ_0(t)$, $LQ_1(t)$, $\ldots LQ_N(t)$ be the quality of the links used by each flow in slot t such that $m \leq LQ_i(t) \leq 1$ for all flows F_i ($F_i \in \mathcal{F}$) and all slots t ($t \in \mathbb{N}$). Under these assumptions, the reliability $R_{i,t}$ of an instance J_i after executing t pulls of the Recorp policy π is lower bounded by $\widehat{R}_{i,t}$.*

Proof. The end-to-end reliability of flow instance J_i after t slots is

$$R_{i,t} = \boldsymbol{P}_t \boldsymbol{\chi}_i = s_0^T \mathcal{M}_{srv(0)} \mathcal{M}_{srv(1)} \cdots \mathcal{M}_{srv(t)} \boldsymbol{\chi}_i.$$

Since $R_{i,t}$ is a number, we can apply the transpose to obtain

$$\begin{aligned}R_{i,t} &= (R_{i,t})^T \\ &= \boldsymbol{\chi}_i^T \mathcal{M}^T_{srv(0)} \mathcal{M}^T_{srv(1)} \cdots \mathcal{M}^T_{srv(t)} s_0.\end{aligned}$$

We observe that $\boldsymbol{\chi}_i$ is an increasing vector by construction, and by extension, $\boldsymbol{\chi}_i^T$. By Lemma A.6 the following must be true as a result:

$$\begin{aligned} R_{i,t} &= \boldsymbol{\chi}_i^T \mathcal{M}_{srv(0)}^T \mathcal{M}_{srv(1)}^T \cdots \mathcal{M}_{srv(t)}^T s_0 \\ &\geq \boldsymbol{\chi}_i^T \widehat{\mathcal{M}}_{srv(0)}^T \mathcal{M}_{srv(1)}^T \cdots \mathcal{M}_{srv(t)}^T s_0. \end{aligned}$$

As a consequence of Lemma A.5, $\boldsymbol{\chi}_i^T \widehat{\mathcal{M}}_{srv(0)}^T$ is an increasing vector, and therefore we can again apply Lemma A.6 to get the following:

$$R_{i,t} \geq \boldsymbol{\chi}_i^T \widehat{\mathcal{M}}_{srv(0)}^T \widehat{\mathcal{M}}_{srv(1)}^T \cdots \mathcal{M}_{srv(t)}^T s_0.$$

Continuing in this way gives the desired result:

$$\begin{aligned} R_{i,t} &\geq \boldsymbol{\chi}_i^T \widehat{\mathcal{M}}_{srv(0)}^T \widehat{\mathcal{M}}_{srv(1)}^T \cdots \widehat{\mathcal{M}}_{srv(t)}^T s_0 \\ &= \widehat{R}_{i,t}. \end{aligned}$$

□

REFERENCES

[1] 2021. ISA100.11a. Retrieved from https://www.isa.org/isa100/.

[2] 2021. WirelessHART. Retrieved from https://fieldcommgroup.org/.

[3] 2021. WUSTL Wireless Sensor Network Testbed. Retrieved from http://mobilab.wustl.edu/testbed.

[4] Yuvraj Agarwal, Bharathan Balaji, Seemanta Dutta, Rajesh K. Gupta, and Thomas Weng. 2011. Duty-cycling buildings aggressively: The next frontier in HVAC control. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'11)*.

[5] Riccardo Bettati. 1994. *End-to-end scheduling to meet deadlines in distributed systems*. Ph.D. Dissertation. University of Illinois at Urbana-Champaign.

[6] Ryan Brummet, Dolvara Gunatilaka, Dhruv Vyas, Octav Chipara, and Chenyang Lu. 2018. A Flexible retransmission policy for industrial wireless sensor actuator networks. In *Proceedings of the IEEE International Conference on Industrial Internet (ICII'18)*.

[7] Alan Burns, James Harbin, Leandro Indrusiak, Iain Bate, Rob Davis, and David Griffin. 2018. Airtight: A resilient wireless communication protocol for mixed-criticality systems. In *Proceedings of the IEEE International Conference on Embedded and Real-time Computing Systems and Applications (RTCSA'18)*.

[8] Richard Candell, Catherine A. Remley, Jeanne T. Quimby, David R. Novotny, Alexandra E. Curtin, Peter B. Papazian, Galen H. Koepke, Joseph E. Diener, and Mohamed T. Hany. 2017. Industrial wireless systems: Radio propagation measurements. Technical Note (NIST TN)-1951 (2017).

[9] Alberto Cerpa, Jennifer L. Wong, Miodrag Potkonjak, and Deborah Estrin. 2005. Temporal properties of low power wireless links: modeling and implications on multi-hop routing. In *Proceedings of the International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc'05)*. https://doi.org/10.1145/1062689.1062741

[10] Octav Chipara, Chenyang Lu, Thomas C. Bailey, and Gruia-Catalin Roman. 2010. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*.

[11] Octav Chipara, Chenyang Lu, John A. Stankovic, and Gruia-Catalin Roman. 2010. Dynamic conflict-free transmission scheduling for sensor network queries. *IEEE Trans. Mobile Comput.* 10, 5 (2010), 734–748.

[12] Nikolaus Correll, Prabal Dutta, Richard Han, and Kristofer Pister. 2017. Wireless robotic materials. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–6.

[13] Behnam Dezfouli, Marjan Radi, and Octav Chipara. 2017. REWIMO: A real-time and reliable low-power wireless mobile network. *ACM Transactions on Sensor Networks (TOSN)* 13, 3 (2017), 1–42.

[14] Manjunath Doddavenkatappa, Mun Chan, and A.L. Ananda. 2012. Indriya: A low-cost, 3D wireless sensor network testbed. *Lecture Notes Inst. Comput. Sci. Soc.-Info. Telecommun. Eng.* 90 (2012), 302–316. https://doi.org/10.1007/978-3-642-29273-6_23

[15] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust mesh networks through autonomously scheduled TSCH. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'15)*.

[16] O. Durmaz Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi. 2012. Fast data collection in tree-based wireless sensor networks. *IEEE Trans. Mobile Comput.* 11, 1 (2012), 86–99. https://doi.org/10.1109/TMC.2011.22

[17] Ken Ferens, Lily Woo, and Witold Kinsner. 2009. Performance of ZigBee networks in the presence of broadband electromagnetic noise. In *Proceedings of the IEEE Canadian Conference of Electrical and Computer Engineering (CCECE'09)*.

[18] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. 2011. Efficient network flooding and time synchronization with glossy. In *Proceedings of the Conference on Information Processing in Sensor Networks*.

[19] Edgar N. Gilbert. 1960. Capacity of a burst-noise channel. *Bell Syst. Tech. J.* 39, 5 (1960), 1253–1265.

[20] Tao Gong, Tianyu Zhang, Xiaobo Sharon Hu, Qingxu Deng, Michael Lemmon, and Song Han. 2019. Reliable dynamic packet scheduling over lossy real-time wireless networks. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS'19)*.

[21] A. Gonga, O. Landsiedel, P. Soldati, and M. Johansson. 2012. Revisiting multi-channel communication to mitigate interference and link dynamics in wireless sensor networks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'12)*. https://doi.org/10.1109/DCOSS.2012.15

[22] Dolvara Gunatilaka, Mo Sha, and Chenyang Lu. 2017. Impacts of channel selection on industrial wireless sensor-actuator networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'17)*. IEEE, 1–9.

[23] Samira Hayat, Evşen Yanmaz, and Raheeb Muzaffar. 2016. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Commun. Surveys Tutor.* 18, 4 (2016), 2624–2661.

[24] Ozlem Durmaz Incel. 2011. A survey on multi-channel communication in wireless sensor networks. *Comput. Netw.* 55, 13 (2011), 3081–3099. https://doi.org/10.1016/j.comnet.2011.05.020

[25] Ankur Kamthe, Miguel A. Carreira-Perpinán, and Alberto E. Cerpa. 2009. M&M: multi-level Markov model for wireless link simulations. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*.

[26] Bo Li, Lanshun Nie, Chengjie Wu, Humberto Gonzalez, and Chenyang Lu. 2015. Incorporating emergency alarms in reliable wireless process control. In *Proceedings of the ACM/IEEE 6th International Conference on Cyber-Physical Systems (ICCPS'15)*. 218–227.

[27] Bo Li, Lanshun Nie, Chengjie Wu, Humberto Gonzalez, and Chenyang Lu. 2015. Incorporating emergency alarms in reliable wireless process control. In *Proceedings of the International Conference on Cyber-Physical Systems (ICCPS'15)*.

[28] J. P. Lynch, Yang Wang, R. A. Swartz, Kung-Chun Lu, and C. H. Loh. 2008. Implementation of a closed-loop structural control system using wireless sensor networks. *Struct. Control Health Monitor.* 15, 4 (2008), 518–539.

[29] Emerson Process management. 2016. System Engineering Guidelines IEC 62591 WirelessHART.

[30] Sirajum Munir, Shan Lin, Enamul Hoque, S. M. Shahriar Nirjon, John A. Stankovic, and Kamin Whitehouse. 2010. Addressing burstiness for reliable communication and latency bound generation in wireless sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'10)*. https://doi.org/10.1145/1791212.1791248

[31] Wolf-Bastian Pöttner, Hans Seidel, James Brown, Utz Roedig, and Lars Wolf. 2014. Constructing schedules for time-critical data delivery in wireless sensor networks. *Trans. Sensor Netw.* 10, 3 (2014), 1–31.

[32] Ranganathan Prakash and Kendall Nygard. 2010. Time synchronization in wireless sensor networks: A survey. *Int. J. UbiComp* 1 (04 2010). https://doi.org/10.5121/iju.2010.1206

[33] Injong Rhee, Ajit Warrier, Mahesh Aia, Jeongki Min, and Mihail L. Sichitiu. 2008. Z-MAC: A hybrid MAC for wireless sensor networks. *IEEE/ACM Trans. Netw.* 16 (2008).

[34] Abusayeed Saifullah, Dolvara Gunatilaka, Paras Tiwari, Mo Sha, Chenyang Lu, Bo Li, Chengjie Wu, and Yixin Chen. 2015. Schedulability analysis under graph routing in WirelessHART networks. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 165–174.

[35] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. 2010. Real-Time scheduling for WirelessHART networks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'10)*. https://doi.org/10.1109/RTSS.2010.41

[36] Wei Shen, Tingting Zhang, Filip Barac, and Mikael Gidlund. 2013. PriorityMAC: A priority-enhanced MAC protocol for critical traffic in industrial wireless sensor and actuator networks. *IEEE Trans. Industr. Info.* 10, 1 (2013), 824–835.

[37] P. Soldati, H. Zhang, and M. Johansson. 2009. Deadline-constrained transmission scheduling and data evacuation in WirelessHART networks. In *Proceedings of the Annual Enterprise Computing Community Conference (ECC'09)*.

[38] Kannan Srinivasan, Maria A. Kazandjieva, Saatvik Agarwal, and Philip Levis. 2008. The β-factor: Measuring wireless link burstiness. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*.

[39] Rodrigo Teles Hermeto, Antoine Gallais, and Fabrice Theoleyre. 2017. Scheduling for IEEE802.15.4-TSCH and slow channel hopping MAC in low power industrial wireless networks: A survey. *Comput. Commun.* 114 (2017), 84 – 105. https://doi.org/10.1016/j.comcom.2017.10.004

[40] Andrew Tinka, Thomas Watteyne, and Kris Pister. 2010. A decentralized scheduling algorithm for time synchronized channel hopping. In *Proceedings of the International Conference on Ad Hoc Networks*. Springer, 201–216.

[41] Hao-Tsung Yang, Kin Sum Liu, Jie Gao, Shan Lin, Sirajum Munir, Kamin Whitehouse, and John Stankovic. 2017. Reliable stream scheduling with minimum latency for wireless sensor networks. In *Proceedings of the International Conference on Structural Engineering and Construction (SECON'17)*.

[42] Tianyu Zhang, Tao Gong, Song Han, Qingxu Deng, and Xiaobo Sharon Hu. 2018. Fully distributed packet scheduling framework for handling disturbances in lossy real-time wireless networks. In *Proceedings of the IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'18)*.

[43] J. Akerberg, M. Gidlund, and M. Bjorkman. 2011. Future research challenges in wireless sensor and actuator networks targeting industrial automation. In *Proceedings of the 9th IEEE International Conference on Industrial Informatics*. 410–415. https://doi.org/10.1109/INDIN.2011.6034912

Received June 2020; revised March 2021; accepted April 2021